UNIVERSIDAD DE GRANADA
DEPARTAMENTO DE ELECTRÓNICA Y
TECNOLOGÍA DE COMPUTADORES

PROCESAMIENTO DIGITAL DE SEÑALES DE ALTAS PRESTACIONES UTILIZANDO EL SISTEMA NUMÉRICO DE RESIDUOS

TESIS DOCTORAL

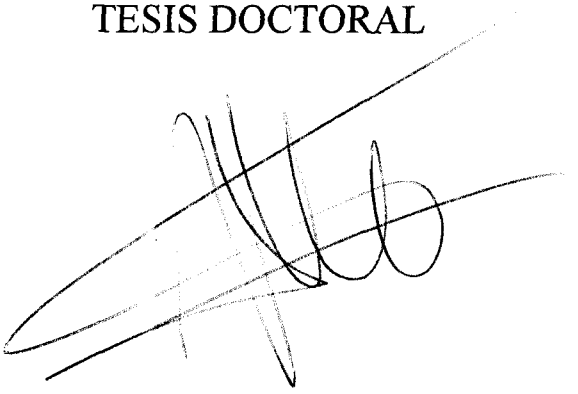
UNIVERSIDAD DE GRANADA
13 MAYO 1999
COMISION DE DOCTORADO

ANTONIO GARCÍA RÍOS

PROCESAMIENTO DIGITAL DE SEÑALES DE ALTAS PRESTACIONES UTILIZANDO EL SISTEMA NUMÉRICO DE RESIDUOS

Antonio García Ríos

TESIS DOCTORAL



UNIVERSIDAD DE GRANADA

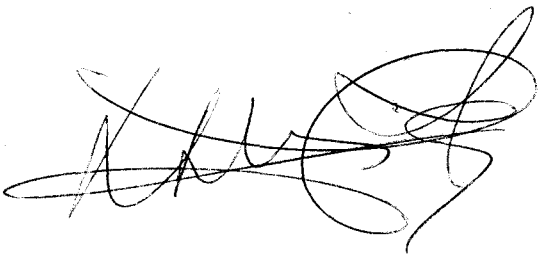
DEPARTAMENTO DE ELECTRÓNICA Y TECNOLOGÍA DE COMPUTADORES

Granada, abril de 1999

D. Antonio Lloris Ruiz, doctor en Ciencias Físicas y Catedrático del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada, como director de Tesis Doctoral de D. Antonio García Ríos

CERTIFICA que la memoria titulada "Procesamiento digital de señales de altas prestaciones utilizando el Sistema Numérico de Residuos" ha sido realizada por D. Antonio García Ríos bajo su dirección, en el Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada. Esta memoria constituye la tesis que D. Antonio García Ríos presenta para optar al grado de Doctor en Ingeniería Electrónica.

Granada, 16 de abril de 1999



Fdo. Antonio Lloris Ruiz

Director de la Tesis

AGRADECIMIENTOS

Deseo expresar mi agradecimiento al profesor Dr. D. Antonio Lloris Ruiz por su dirección y asesoramiento, gracias a los cuales ha sido posible la realización de la investigación recogida en esta memoria. Asimismo quiero agradecer su apoyo y la estrecha colaboración que hemos mantenido durante estos años.

A mis compañeros del Departamento de Electrónica y Tecnología de Computadores, en especial al profesor Dr. D. Luis Parrilla Roure y a D. Mohamed Azimane, por su ayuda, colaboración y amistad.

A los profesores Dra. Anke Meyer-Bäse y Dr. Uwe Meyer-Bäse de la Universidad de Florida, por todo su apoyo y colaboración, y por hacerme sentir entre amigos desde el primer momento.

A los profesores Dr. Fred J. Taylor, de la Universidad de Florida, y Dr. Graham A. Jullien, de la Universidad de Windsor, por su asesoramiento y hospitalidad, y por poner a mi disposición su amplia experiencia y conocimientos, así como los recursos de sus respectivos grupos de investigación. Del mismo modo, quisiera expresar mi agradecimiento a los miembros del High-Speed Digital Architecture Laboratory de la Universidad de Florida y del VLSI Research Group de la Universidad de Windsor, en especial a la Dra. Marjan Shahkarami, por su hospitalidad y simpatía, y al Dr. Binqiao Li por compartir conmigo su amplia experiencia en el desarrollo de circuitos VLSI.

Tampoco puedo dejar de agradecer especialmente a mi familia y a Julia su apoyo constante y su paciencia conmigo.

Finalmente, destacar que este trabajo ha sido financiado en parte a través del Programa de Ayudas para Estancias Breves en el Extranjero concedidas por el Ministerio de Educación y Cultura dentro del Programa Nacional de Formación del Personal Investigador, y por el proyecto de investigación PB96-1397, financiado por el Ministerio de Educación y Cultura a través de la Dirección General de Enseñanza Superior.

A mis padres

A Julia

A Javier

ÍNDICE

ABREVIATURAS Y SIGLAS

A continuación se recogen las abreviaturas y siglas que aparecen a lo largo de esta memoria, a fin de facilitar la lectura de la misma.

ASIC Application Specific Integrated Circuits, circuito integrado de aplicación específica.

BRS Base Removal Scaling, escalado para eliminación de una base.

CIC Cascade Integrator Comb, cascada de integradores y peines.

CIF Caltech Intermediate Form format, formato intermedio del Caltech (California Institute of Technology).

CLK CLocK, señal de reloj en sistemas digitales.

CMC Canadian Microelectronics Corporation, Corporación Canadiense para la Microelectrónica.

CMOS Complementary Metal-Oxide-Semiconductor, proceso metal-óxido-semiconductor complementario.

CPLD Complex Programmable Logic Device, dispositivo lógico programable complejo.

CRT Chinese Remainder Theorem, teorema del resto chino.

DA Distributed Arithmetic, aritmética distribuida.

DFT Discrete Fourier Transform, transformada discreta de Fourier.

DSP Digital Signal Processor, procesador digital de señales (también Digital Signal

Processing, procesamiento digital de señales).

EAB Embedded Array Block, bloque de matrices embebidas (bloque de elementos de memoria en la familia de CPLDs FLEX10K de Altera).

EPLD EPROM technology-based complex Programmable Logic Device, dispositivo lógico programable basado en tecnología EPROM.

EPROM Erasable and Programmable Read Only Memory, memoria de sólo lectura borrable y programable.

FFT Fast Fourier Transform, transformada rápida de Fourier.

FIR Finite Impulse Response, respuesta impulsiva finita.

FPGA Field Programmable Gate Array, matriz de puertas programable por campos.

GDFT Generalized Discrete Fourier Transform, transformada discreta de Fourier generalizada.

GERNS Galois Enhanced Residue Number System, sistema numérico de residuos mejorado con propiedades de los cuerpos de Galois.

GND GrouND, conexión de tierra.

IIR Infinite Impulse Response, respuesta impulsiva infinita.

IOE Input/Output Element, elemento de E/S.

JTAG Joint Test Action Group, grupo para acciones coordinadas sobre test.

LAB Logic Array Block, bloque de elementos lógicos.

LE Logic Element, elemento lógico.

LSI Large Scale Integration, gran escala de integración.

LUT Look-Up Table, tabla de consulta.

MAC Multiply And Accumulate, multiplicar y acumular.

MRC Mixed Radix Conversion, conversión por bases mixtas.

MRRNS Modulus Replication Residue Number System, sistema numérico de residuos

con replicación de módulos.

MSI Medium Scale Integration, media escala de integración.

MSPS MegaSamples Per Second, millones de muestras por segundo.

nETDFF negative Edge-Triggered D Flip-Flop, biestable D disparado por flanco de bajada.

NMOS N-channel Metal-Oxide-Semiconductor, proceso metal-óxido-semiconductor de canal n.

NTT Number Theoretic Transform, transformada numérica teórica.

PCI Peripheral Component Interconnect, estándar para conexión de componentes.

PLA Programmable Logic Array, matriz lógica programable.

PLL Phase-Locked Loop, lazo de fase cerrada.

PMOS P-channel Metal-Oxide-Semiconductor, proceso metal-óxido-semiconductor de canal p.

PROM Programmable Read Only Memory, memoria de sólo lectura programable.

QRNS Quadratic Residue Number System, sistema numérico de residuos cuadrático.

RAM Random Access Memory, memoria de acceso aleatorio.

RISC Reduced Instruction Set Computer, computador de conjunto de instrucciones reducido.

RNS Residue Number System, Sistema Numérico de Residuos.

ROM Read Only Memory, memoria de sólo lectura.

SCR Speed-to-Cost Ratio, relación entre velocidad y coste.

SRAM Static Random Access Memory, memoria de acceso aleatorio estática.

SRNS Symmetric Residue Number System, sistema numérico de residuos simétrico.

TSPC True Single Phase Clock, reloj de fase única.

Vdd Línea de alimentación en circuitos integrados.

VHDL Very high-speed integrated circuit Hardware Description Language, lenguaje para descripción de hardware para circuitos integrados de muy alta velocidad.

VLSI Very Large Scale Integration, escala muy grande de integración.

WFTA Winograd's Fourier Transform Algorithm, algoritmo de Winograd para la transformada de Fourier.

PRÓLOGO

En los últimos años [IBR94, TAT95, KIN97] ha surgido un gran interés por la implementación de sistemas de procesamiento de altas prestaciones basados en el uso del Sistema Numérico de Residuos [SZA67] (RNS: Residue Number System); si bien su simplicidad y capacidad teórica para agilizar las aplicaciones más frecuentes de procesamiento digital de señales resultaron un aliciente para su estudio [TAY84], las limitaciones tecnológicas unidas a los requerimientos especiales del RNS han limitado su aplicación hasta hace pocos años. Sin embargo, los recientes avances en la integración VLSI y el desarrollo de dispositivos programables han permitido que el RNS haya pasado de una mera especulación teórica a una alternativa factible en ciertas aplicaciones; más aún, el RNS se ha mostrado recientemente como un claro competidor a las arquitecturas de cómputo tradicionales para el procesamiento digital de señales en un gran número de aplicaciones.

En este trabajo se desarrollan estructuras de cálculo en el Sistema Numérico de Residuos orientadas a niveles superiores de aplicación y diseño, considerando las operaciones más comunes en la aritmética de residuos y en las aplicaciones en las que se puede encontrarla de manera usual. Como colofón, esta memoria recoge algunas de las más novedosas propuestas que se pueden encontrar en la literatura para conectarlas con otros esquemas propuestos durante los años 80, a fin de fundir lo mejor de ambos y llevarlo a la práctica. Es este un trabajo que deriva del interés que últimamente ha suscitado el RNS, motivado principalmente por la necesidad de un procesamiento numérico de altas prestaciones, a una velocidad superior a la ofrecida por la tecnología actual con los esquemas convencionales; evidentemente, a la vista de los avances de la microelectrónica digital (capaz de ofrecer ya procesadores de propósito general de

altísimas prestaciones), estas necesidades se centran en aplicaciones de carácter muy específico y orientadas en la mayoría de los casos al procesamiento de señales en tiempo real.

Esta memoria recoge las aportaciones realizadas por el doctorando en el Departamento de Electrónica y Tecnología de Computadores durante los últimos años, que han seguido tres grandes líneas:

- desarrollo de nuevas propuestas para la realización de operaciones aritméticas básicas en el RNS: suma, multiplicación y escalado;
- desarrollo de aplicaciones basadas en el RNS haciendo uso de dispositivos lógicos programables de última generación;
- desarrollo de nuevas alternativas para la integración VLSI de aplicaciones basadas en el RNS, con especial atención al diseño orientado al bajo consumo de potencia.

La memoria está estructurada tal y como se describe a continuación:

CAPÍTULO I: Introducción. Este primer capítulo expone las ideas básicas que motivan el trabajo realizado, así como la relevancia adquirida durante los último años por el RNS, su especial interés al considerar las más modernas familias de dispositivos programables y las líneas maestras en el desarrollo de esta memoria.

CAPÍTULO II: Fundamentos del RNS. Se expondrán en primer lugar los conocimientos mínimos necesarios sobre el RNS para el seguimiento de los restantes capítulos, así como ejemplos de algunas de las aplicaciones clásicas de la aritmética de residuos; éstas darán una visión de conjunto de los diferentes esquemas existentes y las posibilidades del RNS.

CAPÍTULO III: Módulos aritméticos para RNS. Este capítulo tiene por objeto sentar las bases de las realizaciones físicas que se expondrán en el capítulo IV, al tiempo que muestra las metodologías clásicas para la implementación de la suma y la multiplicación en módulo.

CAPÍTULO IV: Multiplicadores para módulos primos con cauce segmentado. Haciendo uso de las ideas expuestas en el capítulo anterior se mostrará aquí el diseño concreto de una unidad multiplicadora módulo p de cauce segmentado y la posibilidad de detección de errores sin coste adicional, mostrando como ejemplo el caso de p=127; como se verá, su diseño es extremadamente sencillo, y dará una idea de la potencialidad del Sistema Numérico de Residuos.

CAPÍTULO V: Escalado en el RNS. En este capítulo se propondrán varias alternativas para realizar la operación de escalado en el RNS, ya que ésta es imprescindible en muchas aplicaciones del RNS, mostrando las mejoras conseguidas frente a las propuestas de otros autores. En concreto, se mostrarán dos alternativas diferentes: un esquema basado en módulos aritméticos convencionales, aprovechando los resultados del capítulo IV, y un nuevo esquema de escalado completamente basado en el uso de tablas de consulta (LUT: look-up table) y con una latencia de dos ciclos de acceso a memoria; además, presentaremos una implementación VLSI de este último algoritmo.

CAPÍTULO VI: Desarrollo de aplicaciones RNS sobre dispositivos programables. Haciendo uso de algunas de las ideas aparecidas en los capítulos previos, se sentarán las bases para el diseño de aplicaciones basadas en el RNS haciendo uso de los dispositivos programables de última generación; en concreto, se expondrá la realización RNS de una arquitectura CIC de tres etapas y las ventajas que presenta ésta frente a la implementación clásica basada en aritmética binaria en complemento a dos. Por otra parte, se presentará una nueva arquitectura basada en la fusión del RNS y la aritmética distribuida, y se mostrarán sus ventajas sobre las correspondientes implementaciones binarias.

CAPÍTULO VII: Nuevas estrategias de sincronización VLSI para aplicaciones RNS. Este capítulo recoge las aportaciones realizadas en el campo de la integración VLSI, y más en concreto, una nueva estrategia de sincronización capaz de reducir los picos de corriente en las líneas de alimentación de circuitos integrados aprovechando las características del RNS. Esta estrategia es aplicable a cualquier arquitectura formada por un elevado número de canales independientes que trabajen en paralelo.

CAPÍTULO VIII: Conclusiones y principales aportaciones. En el último capítulo de esta memoria se tratarán de extraer y condensar las ideas más destacadas que aparecen a lo largo del trabajo, detallando las principales aportaciones realizadas en el campo del RNS, así como las líneas a seguir en investigaciones futuras.

Al final de la memoria se incluyen diversos apéndices con los siguientes contenidos:

APÉNDICE A: Introducción al álgebra de los cuerpos finitos. Este apéndice describe los conceptos algebraicos básicos referentes a los cuerpos finitos o de Galois, GF(p), estructuras algebraicas empleadas en el desarrollo de parte del trabajo expuesto.

APÉNDICE B: Familia de dispositivos lógicos programables FLEX10K de Altera. Esta familia ha sido empleada para desarrollar parte de la investigación recogida en este memoria, de forma que esta apéndice describe la arquitectura básica y los aspectos más destacados de estos dispositivos.

APÉNDICE C: Proceso CMOS14TB. Dado que éste ha sido el proceso empleado para el diseño de las estructuras VLSI mostradas en esta memoria, así como la obtención de los resultados del capítulo VII, este apéndice recoge algunos de los aspectos más destacados de este proceso para la fabricación de circuitos integrados con tecnología CMOS.

Para finalizar el prólogo, sólo realizar algunas indicaciones para facilitar la lectura de esta memoria:

- La memoria, tal y como hemos visto, está dividida en diferentes capítulos; estos capítulos, a su vez, constan de diferentes secciones, que numeraremos con dos dígitos, uno referente al capítulo y otro a la sección, y así serán referidos a lo largo de la memoria. Ejemplo: sección 2.3.
- Cuando sea necesario, las secciones se dividirán en apartados, que llevarán los mismos dígitos que los correspondientes a la sección en la que se encuadran, acompañados de otro más que se referirá al orden de aparición del apartado dentro

de dicha sección. Ejemplo: apartado 3.4.1.

- Las figuras y tablas están numeradas por capítulos, con un primer dígito que señala éste, y otro que se refiere al elemento en sí. Ejemplos: figura 2.1, tabla 6.2.
- Las expresiones matemáticas y ecuaciones son referenciadas, de manera similar, con dos dígitos, el primero de ellos referente al capítulo en el que aparecen. Ejemplo: ecuación (4.1).
- Por último, las referencias bibliográficas aparecen designadas en el texto con tres letras, correspondientes al primer autor, y dos cifras, que indican el año de publicación; en caso de existir varias referencias con la misma denominación, ésta irá acompañada de una letra al final. Las referencias se hallan ordenadas alfabéticamente al final de esta memoria. Ejemplo: [TAY84], [GAR98a], [GAR98b].

CAPÍTULO I

INTRODUCCIÓN

En este primer capítulo se introducirá el tema objeto de esta memoria, exponiendo los principales motivos del interés surgido por el RNS para la implementación de sistemas de altas prestaciones durante los últimos años. Si en un principio su uso se vio limitado por sus excesivos requerimientos, los últimos avances tecnológicos han permitido considerar al RNS como una alternativa más frente a los esquemas aritméticos convencionales y los sistemas basados en microprocesadores de propósito general. A pesar de las limitaciones inherentes al RNS, su modularidad y su gran capacidad para la realización rápida de suma, resta y multiplicación le confieren una especial aplicabilidad en el campo del procesamiento de señales, donde la mayoría de algoritmos se basan en la realización de un elevado número de estas operaciones.

1.1. EL SISTEMA NUMÉRICO DE RESIDUOS

El estudio del Sistema Numérico de Residuos (RNS: Residue Number System) se remonta al siglo III [TAY84], en la antigua China; en el libro *Suan-ching* de Sun Tzu aparecen los versos:

Hay cosas de las cuales no conocemos el número
Si las contamos por tríos, nos quedan 2.
Si las contamos por quintetos, nos quedan 3.
Si las contamos por septetos, nos quedan 2.
¿Cuántas cosas hay?
La respuesta, 23.

En este libro se recoge el procedimiento para poder concluir que la respuesta es 23, con una fórmula para manipular los restos de las divisiones por 3, 5 y 7 de un entero. Es por este motivo (sección 2.1) por el que una de las reglas básicas de la aritmética de residuos se conoce como el *teorema del resto chino* (CRT: Chinese Remainder Theorem). Este teorema, y el resto de la teoría de residuos, fue retomado en el siglo XIX por Carl Gauss en sus *Disquisiciones Aritméticas*.

El primer intento en el siglo XX de emplear las propiedades del RNS corresponde a D. H. Lehmet, quien en 1932 construyó una máquina electromecánica para factorizar los números de Mersenne [LIP81], a la que llamó criba fotoeléctrica. Posteriormente, a mediados de la década de los 50, los checos Svaboda y Valach diseñaron una máquina cableada para el Sistema Numérico de Residuos, que emplearon en el estudio de códigos de error; la misma idea fue desarrollada por Aiken y Garner [GAR59].

Desde el final de los años 50 hasta mediados de los 60, el Departamento de Defensa de los Estados Unidos financió el trabajo de Szabo y Tanaka [SZA67] sobre el RNS en la empresa Lockheed; trabajaron en la construcción de un correlador digital de propósito específico, mientras otro equipo lo hacía en el diseño de una máquina de propósito general. En el campo práctico estas investigaciones encontraron poco éxito, debido a la cantidad de memoria requerida por diseños especializados; el resultado más tangible fue el libro escrito por Szabo y Tanaka [SZA67], una de las publicaciones clásicas en el campo del Sistema Numérico de Residuos. La causa de este desesperanzador rendimiento fue que la tecnología del momento no podía satisfacer las demandas que exigía el Sistema Numérico de Residuos.

Sin embargo, desde mediados de los 70, tecnología y teoría han ido convergiendo y ahora es fácil implementar la memoria requerida por circuitos que usen el RNS, e incluso se está trabajando en realizaciones electro-ópticas, aunque esto no es, por ahora, más que un campo de investigación. A pesar de todo, el RNS está encontrando su lugar en muchas aplicaciones en las que intervienen filtros digitales y transformadas, tales como implementaciones de la FFT a alta velocidad [JUL88], sistemas de corrección de errores [ORT92, KAT96], filtrado [IBR94, GAR98a], etc.

1.1.1. Interés de la aritmética de residuos

Actualmente, el avance tecnológico de la microelectrónica ha hecho que el procesamiento digital presente indudables ventajas frente al analógico; podemos hoy día encontrar diferentes sistemas de procesamiento digital en aplicaciones que van desde el ámbito doméstico (multimedia, sonido de alta calidad, imagen) al tratamiento avanzado de señales (compresión de voz e imágenes, transmisión de datos, telefonía móvil, HDTV, etc.). En concreto, el procesamiento digital de señales se refiere al tratamiento de las mismas con transformadas, filtros, etc., que generalmente se llevan a cabo con algoritmos de la forma:

$$y_n = \sum_{k=1}^{N} \alpha_{n-k}\, x_k \qquad \textbf{(1.1)}$$

donde α_n son coeficientes conocidos *a priori*, y $\{x_n\}$ e $\{y_n\}$ son las secuencias de muestras de entrada y salida, respectivamente. De este modo, sistemas que pueden necesitar un complicado diseño analógico que, en muchos casos, es preciso estudiar a través de aproximaciones del comportamiento de los diferentes componentes, se realizan a través de operaciones suma y producto en un sistema digital; éste puede ir desde un sencillo microprocesador de 8 bits a complejos DSPs de arquitectura RISC, pasando por otros sistemas digitales más específicos. Por tanto, realizando las oportunas conversiones A/D y D/A de las señales de entrada y salida, respectivamente, se pueden obtener importantes ventajas frente al procesamiento analógico tradicional:

- Los diseños analógicos, dada la complejidad de la descripción de los diferentes elementos y, por tanto, de las ecuaciones que describen los sistemas empleados, hacen difícil la construcción de procesadores analógicos perfectamente ajustados a las especificaciones requeridas; esto provoca la inserción de numerosos puntos de ajuste en los diseños analógicos de gran complejidad. Un tratamiento digital del problema puede evitar estos ajustes.

- Por su propia naturaleza, la complejidad de los sistemas analógicos está limitada; a medida que aumenta el orden del sistema, se incrementan la complejidad del diseño y la dificultad de ajuste, al tiempo que el hardware requerido es cada vez mayor. Por el contrario, si la misma aplicación se lleva a cabo sobre un microprocesador de propósito general o un DSP, el mismo hardware puede realizar tareas de muy diversa complejidad sin más que alterar el programa que ejecuta.

No obstante, existe una razón muy importante para que un tratamiento digital no siempre sea posible; esta no es otra que la velocidad de procesamiento, que limita el rango de frecuencias de las señales a tratar (recordemos que para operar correctamente la frecuencia de muestreo ha de ser al menos el doble de la máxima componente de la señal, por lo que es preciso tratar digitalmente dos muestras en cada período de muestreo). Este problema se intenta resolver a través de dos vías:

- mejorar la tecnología a fin de dotar de una mayor velocidad de operación a los sistemas existentes, tales como microprocesadores, etc.;

- buscar esquemas alternativos a las técnicas aritméticas usuales para la realización de procesadores ASIC (Application Specific Integrated Circuit), dado que el procesamiento digital es básicamente un problema de cálculo.

En esta última alternativa es donde el RNS [GAR96] entra en juego; en los esquemas de representación y cálculo convencionales, la principal fuente de retardo en la operación de suma se debe a la propagación del acarreo entre las diferentes etapas, lo que intenta resolverse con circuitos especiales [WAS82], como los sumadores de acarreo

anticipado (carry look-ahead adders) o los generadores de acarreo, que significan un incremento sustancial del hardware empleado; además, la multiplicación se lleva a cabo con módulos sumadores, tanto en realizaciones secuenciales que intentan mejorar el algoritmo usual de lápiz y papel, como el algoritmo de Booth [WAS82] o generación de productos parciales, como en realizaciones puramente combinacionales (arrays de celdas, siendo cada celda un módulo cercano a un sumador binario). Precisamente, la principal propiedad del Sistema Numérico de Residuos se centra en que es un sistema de representación sin propagación de acarreo entre canales (v. sección 2.1), con lo que se puede operar a una velocidad sensiblemente superior a la obtenida en otros esquemas [GAR98a].

Por otra parte (v. sección 2.1), las operaciones en el sistema de residuos se llevan a cabo de manera paralela sobre un conjunto de módulos, cada uno de ellos menor que el rango dinámico del sistema total, con lo que la velocidad de operación vendrá limitada por la menor velocidad de operación de todos los módulos; por tanto, podemos trabajar con el RNS sobre grandes rangos sin penalizar la velocidad de operación, ya que el rango se amplía sin más que añadir nuevos módulos al RNS considerado. Esto solventa otra de las desventajas del procesamiento digital, como es la precisión limitada por la misma discretización del rango de las señales empleadas; generalmente, en los esquemas de cómputo convencionales, precisión y velocidad son parámetros antagónicos, dado que la propagación del acarreo ralentiza la velocidad de funcionamiento a medida que aumentamos el rango empleado.

Este sistema numérico con más de 17 siglos de antigüedad ha sido objeto de gran interés en los últimos años, ya que los sistemas digitales con unidades de aritmética de residuos pueden desempeñar un importante papel en sistemas de alta velocidad y tiempo real que soportan el procesamiento paralelo de datos enteros. Esto es así ya que es un sistema de numeración con una propagación de acarreo muy limitada y que permite llevar a cabo las operaciones de suma, resta y multiplicación de forma paralela; así se elimina una de las principales fuentes de retardo de los circuitos aritméticos, la gestión de la información sobre el acarreo. De este modo, una operación básica en control de procesos en tiempo real, procesamiento de señales y de imágenes, gráficos, etc., como es la

multiplicación, puede llevarse a cabo de una forma bastante más rápida que con los sistemas de numeración usuales.

1.2. PERSPECTIVA ACTUAL DE LA ARITMÉTICA DE RESIDUOS

Como ya se ha citado anteriormente, el RNS ha suscitado gran interés desde hace varias décadas, pero no siempre se ha podido trasladar fácilmente a realizaciones prácticas; en concreto, veremos la realización de ciertas operaciones a través de memorias ROM dispuestas como tablas (look-up table), de forma que el resultado de dichas operaciones se halla almacenado en dichas tablas; sin embargo, hasta hace pocos años, el coste y capacidad de las memorias disponibles hacía inviable este tipo de realización, dado el coste desorbitado. Hoy día, el avance de la tecnología microelectrónica permite disponer de la cantidad necesaria de memoria con un coste razonable para pensar en la realización práctica de la aritmética de residuos, sobre todo teniendo en cuenta el carácter especial de muchas de las aplicaciones (v. sección 2.3).

1.2.1. Nuevas tendencias utilizando dispositivos programables

En los últimos años ha surgido una serie de dispositivos lógicos programables, tales como EPLDs (EPROM technology-based complex Programmable Logic Devices) y FPGAs (Field Programmable Gate Arrays), que han despertado un enorme interés por sus posibilidades para el desarrollo de sistemas digitales; en concreto, las FPGAs fueron introducidas en 1985 por Xilinx, Inc., y en la actualidad, un gran número de compañías fabrican una amplia variedad de este tipo de dispositivos: Actel, Altera, Xilinx, etc. La filosofía de las FPGAs se basa en disponer de un chip modificable o reconfigurable, de manera que un cambio de las especificaciones o de la funcionalidad requerida en nuestro sistema no implica la sustitución de dicho chip; en concreto, las FPGAs se configuran usualmente desde una memoria en el proceso de arranque, de manera que sólo hay que sustituir dicha memoria, o modificar su contenido, para variar las funciones que se realizan en el chip [ALT98]. Una FPGA está formada por una serie de bloques lógicos

configurables (CLB: Configurable Logic Block, para Xilinx; LE: Logic Element, para Altera, etc.), bloques que van desde una pareja de transistores hasta pequeñas tablas de consulta de 16×1 bits, pasando por puertas NAND o biestables tipo D, según los diferentes fabricantes. Además, gran parte del chip está dedicado a las interconexiones entre los diferentes conjuntos de elementos lógicos, que también son programables, junto a los bloques necesarios para proporcionar E/S disponibles para el usuario. Lo que es más, el continuo avance tecnológico ha propiciado en los últimos años la aparición de los CPLDs (Complex Programmable Logic Devices), como las familias FLEX8000 y FLEX10K de Altera, con elementos lógicos de mayor complejidad que los presentes en las FPGAs. Por tanto, este tipo de dispositivos es ideal para la construcción de prototipos, al mismo tiempo que permite una flexibilidad enorme en el diseño; de este modo, si todo nuestro sistema está contenido en uno de estos chips, sólo hay que cambiar la configuración del mismo para modificar, por ejemplo, el rango del RNS utilizado en un multiplicador, sin más que alterar (v. sección 3.2.3) el contenido de las diferentes tablas almacenadas en el chip, y sobre las que se realizan las operaciones.

De este modo, los dispositivos programables proporcionan:

- diseño y verificación más rápidos: el diseño puede cargarse *on-line* sobre un sistema de prueba para su verificación, sin necesidad de esperar la fabricación de prototipos o de trabajar sobre simulaciones por ordenador;

- diseños modificables sin penalización: dado que los dispositivos se configuran a partir del contenido de una memoria o un fichero, una modificación puede realizarse en pocas horas y con mucho menos riesgo que en el caso de dispositivos tradicionales;

- todo esto hace que nuestro producto pueda llegar antes al mercado, factor que es cada día más fundamental en un sector tan competitivo y en constante desarrollo como es el de la microelectrónica digital.

Por otra parte, la presencia de un biestable en la salida de cada uno de los

elementos lógicos en la mayoría de familias de dispositivos programables redunda en una mayor facilidad para la inclusión de segmentación de cauce sin coste adicional; de este modo, es posible segmentar prácticamente cualquier sistema realizable sobre dispositivos programables sin necesidad de nuevos recursos, lo que se traduce en una mayor eficiencia y velocidad.

En lo que se refiere a las ventajas del uso del RNS sobre dispositivos programables, la capacidad limitada de éstos ha supuesto hasta fechas recientes un grave inconveniente para la realización de aplicaciones basadas en el RNS. Sin embargo, tal y como hemos mencionado anteriormente, la aparición de nuevas familias de CPLDs como la FLEX10K [ALT98], que incluyen pequeñas tablas dedicadas de 256×8 bits configurables como ROM o RAM, ha supuesto un gran aliciente para el uso del RNS; de hecho, y tal y como venimos comentando, la necesidad de incluir un número considerable de tablas de consulta en sistemas basados en el RNS es una de las principales desventajas de éstos; más aún, la implementación de estas tablas usando los elementos lógicos de una FPGA supone un elevado coste. De este modo, la posibilidad de incluir estas tablas sin necesidad de emplear elementos lógicos supone una ventaja considerable, especialmente en aplicaciones basadas en el RNS.

1.2.2. Uso del RNS en circuitos ASIC VLSI

El otro gran campo de aplicación del RNS ha sido el diseño de circuitos ASIC, a partir de la aparición de los circuitos VLSI (Very Large Scale Integration). Como se viene comentando a lo largo de este capítulo introductorio, las necesidades especiales del RNS limitaron en un principio su aplicación práctica; posteriormente, dispositivos MSI (Medium Scale Integration) y LSI (Large Scale Integration) hicieron posible el desarrollo de aplicaciones basadas en componentes discretos [JEN77, HUA81]. Sin embargo, la llegada de los dispositivos VLSI ha supuesto un nuevo impulso para el desarrollo de sistemas basados en el RNS, con la posibilidad de incluir en un solo chip sistemas completos, lo que ha propiciado la aparición de numerosas implementaciones [TAY82, WAN94, SMI95] de sistemas haciendo uso del RNS; además, se ha empleado un gran

número de arquitecturas diferentes para la realización de dichos sistemas, desde diseños específicos para el RNS [WIG94] a arquitecturas sistólicas [BAN94, SMI95] pasando por autómatas celulares [YOR91] u otras alternativas.

Tradicionalmente, el uso del RNS en ASICs ha supuesto una mejora en las prestaciones de la arquitectura propuesta en comparación con la alternativa binaria. Sin embargo, el espectacular incremento de prestaciones en la tecnología microelectrónica hace que la velocidad de procesamiento requerida para muchas aplicaciones no exija el uso de esquemas especiales como el RNS para su consecución; pero si el incremento de velocidad en arquitecturas convencionales gracias al avance tecnológico ha quitado al RNS parte de su potencial, este mismo aumento de prestaciones ha propiciado la aparición de nuevos problemas en las arquitecturas convencionales, tales como la distribución de reloj, consumo de potencia, etc. Y en estos nuevos problemas es donde el RNS ha encontrado un nuevo campo de aplicación, ya que características como modularidad o independencia entre canales hacen del RNS una útil herramienta para paliar éstos y otros problemas sin perjudicar la velocidad del sistema.

1.3. CONCLUSIÓN

En este primer capítulo se ha realizado una revisión de la evolución histórica del RNS, al mismo tiempo que se han expuesto los motivos que provocan el interés en el desarrollo de aplicaciones basadas en este sistema numérico con más de 17 siglos de antigüedad. Por otra parte, se han detallado las líneas de investigación abiertas en este campo: desarrollo sobre dispositivos programables y mejora de propuestas ASIC para integración VLSI, que serán desarrolladas en los capítulos sucesivos de esta memoria.

CAPÍTULO II

FUNDAMENTOS DEL RNS

En este capítulo se exponen las ideas básicas para el desarrollo posterior del trabajo, incluyendo una introducción a las operaciones más comunes en la aritmética de residuos, tales como suma, resta, multiplicación y escalado; además, se introducen algunas otras características del sistema de residuos, tales como la posibilidad para detección y corrección de errores, nociones sobre la conversión binario-RNS, etc. Por otra parte, se presentan algunas aplicaciones clásicas del RNS como ejemplo del potencial de este sistema numérico.

2.1. INTRODUCCIÓN AL SISTEMA NUMÉRICO DE RESIDUOS

A continuación se introducirán los conceptos básicos acerca del RNS, con una descripción de su funcionalidad y los aspectos más destacados relacionados con el mismo.

2.1.1. Operaciones básicas en el RNS

Los sistemas numéricos usuales son lineales, posicionales y pesados, es decir, todas las posiciones derivan su peso de una misma base. El RNS, en cambio, emplea bases posicionales que son primas entre sí, como, por ejemplo, 5, 3 y 2. De este modo, tal como lo hace el número binario 111, en el RNS el trío $[R_5, R_3, R_2]$ determina con las bases

anteriores el número decimal 7, donde:

$$R_5 = 7 \bmod 5 = 2 \qquad R_3 = 7 \bmod 3 = 1 \qquad R_2 = 7 \bmod 2 = 1$$

Esto queda formalizado a través del siguiente teorema, conocido como *teorema del resto chino*:

Teorema 2.1: *Dado un conjunto de números primos entre sí,* $m_1, m_2, \ldots, m_n$, *entonces para todo* $X<M$ *el conjunto de residuos* $\{X \bmod m_i \mid 1 \le i \le n\}$ *es único, siendo*

$$M = \prod_{i=1}^{n} m_i \qquad \textbf{(2.1)}$$

Demostración: la prueba es directa; supongamos que existen dos números Y y Z tales que tengan la misma representación en residuos, de forma que:

$$y_i = Y \bmod m_i$$
$$z_i = Z \bmod m_i$$

e $y_i = z_i$ para todos los m_i. Por tanto, Y-Z es múltiplo de cada m_i, y también del mínimo común múltiplo de los m_i; pero, dado que los m_i son primos entre sí, su mínimo común múltiplo es M, con lo que Y-Z sería múltiplo de M, e Y y Z no podrían ser simultáneamente menores que M, q.e.d.

De este modo, el RNS definido por los módulos $\{m_1, m_2, \ldots, m_n\}$ permite representar M enteros diferentes en el rango $[0, M\text{-}1]$, formando un grupo cíclico. Precisamente de este hecho deriva el interés por el Sistema Numérico de Residuos, ya que la aritmética queda definida sobre el anillo (v. Apéndice A) de los enteros módulo M, y éste, a su vez, es isomórfico a la suma directa de los anillos correspondientes a los diferentes módulos que definen el RNS; por tanto, si $\square$ representa suma, resta o multiplicación, y $0 \le X, Y < M$, entonces se satisface:

$$Z = (X \square Y) \bmod M \equiv [z_1, z_2, \ldots, z_n]$$
$$z_i \equiv (x_i \square y_i) \bmod m_i \quad (i = 1, 2, \ldots, n) \tag{2.2}$$

Por tanto, la aritmética en el RNS se realiza sobre n canales independientes de rango dinámico menor que el definido por el conjunto de los mismos; así, no hay propagación de acarreos entre cada uno de los residuos que representa a cada número, y la aritmética es cerrada en cada una de estas posiciones. Esto es lo que permite realizar sumas y multiplicaciones de números grandes a la misma velocidad que con números pequeños, ya que la velocidad está determinada por el canal más lento. Como ejemplo se muestran algunas operaciones en el RNS definido por {5, 3, 2}, con M=30:

17	[2, 2, 1]	8	[3, 2, 0]
$\|+\|_{30}$ 16	$\|+\|_{30}$ [1, 1, 0]	$\|+\|_{30}$ 19	$\|+\|_{30}$ [4, 1, 1]
3	[3, 0, 1]	27	[2, 0, 1]

de forma que cada pareja de residuos se suma de forma independiente, eliminando cualquier acarreo entre posiciones. Para la multiplicación:

7	[2, 1, 1]
$\|\times\|_{30}$ 6	$\|\times\|_{30}$ [1, 0, 0]
12	[2, 0, 0]

y de nuevo, cada columna se multiplica de forma independiente.

Respecto a la resta, dado que (a mod m) - (b mod m) = (a-b) mod m, esta operación no presenta ningún problema en aritmética de residuos y las operaciones en los diferentes módulos se llevan a cabo sobre números positivos; además, para representar números negativos sólo es necesaria una traslación del rango representado por el RNS; así, los enteros $0 \leq X < M/2$ representan su propio valor, mientras que $X \geq M/2$ representan a los negativos, de forma que el rango [-M/2, M/2] se hace corresponder con los enteros [0, M-1]. Esto es equivalente a:

$$x_i = \begin{cases} X \bmod m_i & \text{si } X \geq 0 \\ (M - |X|) \bmod m_i & \text{en otro caso} \end{cases} \quad \textbf{(2.3)}$$

Otra alternativa es el RNS simétrico (SRNS: Symmetric Residue Number System), donde el residuo x_i de X respecto de uno de los módulos m_i es un entero con signo en el conjunto $\{-(m_i-1)/2,..., 0,..., (m_i-1)/2\}$, y el número de enteros representables de manera única sigue siendo:

$$M = \prod_{i=1}^{n} m_i$$

Cualquier entero $X \in [0, M-1]$ se codifica de manera única en el SRNS con la n-tupla $[x_1, x_2,..., x_n]$, siendo:

$$x_i = |X|_{m_i} = X - \left\lfloor \frac{X}{m_i} \right\rfloor m_i \quad \textbf{(2.4)}$$

Como ya se mencionó antes, la velocidad a la que se puede operar viene determinada sólo por el más lento de los diversos canales en los que se realiza la operación para cada módulo; por tanto, es preciso buscar en la elección de los módulos un compromiso entre la velocidad, rango dinámico a representar y recursos a utilizar.

Otra cuestión que entra en juego en la elección de los diferentes módulos m_i es la existencia o no de circuitos aritméticos convencionales para estos módulos. A este respecto, y dadas las aplicaciones de la representación en residuos, es preciso realizar las operaciones a la mayor velocidad posible, por lo que la sustitución de la lógica tradicional por memorias [BAY87b] en las que se almacena el resultado de la operación para cada combinación de entradas es una alternativa muy interesante, cada día más, a medida que aumenta la velocidad y el nivel de integración de los diferentes dispositivos de memoria; de esta forma, y como ya se ha comentado, las operaciones que implica el RNS van adquiriendo mayor importancia a medida que aumenta la capacidad de memoria que facilita la tecnología actual.

2.1.2. Otras operaciones en el RNS

En los sistemas convencionales cada cifra tiene un peso perfectamente determinado, lo que permite comparar dos números haciéndolo dígito a dígito; del mismo modo, es fácil la detección de signo, sin más que comprobar, en base 2 y para las diferentes formas de representación (signo y magnitud, y complemento a uno y a dos) el valor del bit más significativo. Desgraciadamente, esto no es posible en el Sistema Numérico de Residuos; no caben afirmaciones del tipo $[1, 0, 4] \geq 10$ con los submódulos $\{3, 4, 5\}$. Esta dificultad para la comparación de magnitudes y la detección de signo limitan las aplicaciones del RNS frente a las arquitecturas aritméticas tradicionales y lo restringe a aplicaciones en las que se requiere un elevado número de sumas y productos, como se verá más adelante. Además, dado que la división es una sucesión de restas y comparaciones, es una operación difícil de implementar en el Sistema Numérico de Residuos; lo que es más, dado que es un sistema de representación de enteros, el RNS no es cerrado para la división. Banerji propuso en 1981 [BAN81] una rutina de división aproximada a partir de la división estándar de enteros; más recientemente se han propuesto otros algoritmos de división en el RNS [HIT95, POS96, HIA97], pero la complejidad de esta operación es aún mucho mayor que la de suma o multiplicación.

Una cuestión relacionada con esto último es la detección de desbordamiento (overflow); en realidad, no existe desbordamiento en el RNS, ya que la aritmética queda definida sobre un grupo cíclico; sin embargo, cuando el RNS es utilizado para la resolución de algoritmos del tipo de la ecuación (1.1), es preciso asegurar que ningún resultado parcial supere el rango dinámico máximo, ya que esto conduciría a un resultado final erróneo. Si en los sistemas convencionales la detección de desbordamiento se realiza fácilmente inspeccionando el valor del dígito más significativo, en la representación con residuos habría de llevarse a cabo como comparación de magnitud, que como hemos visto, es una operación compleja; este problema ha llevado a la introducción del escalado para evitar el desbordamiento durante el funcionamiento del sistema. El escalado es la composición de una variable con una constante de valor conocido; esto hace del escalado una forma especial de división, lo que representa un problema importante para su

implementación. Sin embargo, el hecho de que el divisor sea constante simplifica el problema hasta el punto de hacerlo abordable en determinadas aplicaciones. Jullien ha propuesto un método basado en la utilización de ROM [JUL78], que es el más utilizado hasta ahora; en concreto, su trabajo recoge dos propuestas: la primera consiste en modificar iterativamente el número a fin de conseguir una división exacta que proporcione un resultado truncado o redondeado; la segunda de las propuestas supone una mejora respecto del hardware necesario y se basa en la suma de ciertos vectores, estimados según una métrica, para obtener el número escalado; sin embargo, en esta última propuesta la cota de error depende del número de residuos que haya que escalar y no es fácil de someter a un tratamiento general.

Otro problema que se plantea con el uso del RNS es el de la corrección de errores, al no existir el concepto de dígitos más o menos significativos; por tanto, la pérdida de un dígito, o un bit, puede alterar sustancialmente el significado decimal de la representación en residuos, mientras que en los sistemas pesados convencionales el perjuicio será función del peso correspondiente al bit alterado. Todo esto hace que para ciertas aplicaciones sean necesarios esquemas de detección y corrección de errores que operen a una velocidad que no afecte a la capacidad de trabajo en tiempo real del sistema de residuos. En 1972, Mandelbaum [MAN72] propuso un método para la corrección de errores simples en un sistema de residuos empleando dos módulos redundantes (redundancia triple); este método se ha extendido con posterioridad a errores múltiples. Básicamente, consiste en el uso de paridad bidimensional, de manera que cada residuo se codifica incluyendo un bit de paridad, al tiempo que se añade una palabra extra en la que su bit j-ésimo es el bit de paridad que se obtiene con los bits j-ésimos de las palabras correspondientes al residuo en cada módulo; el formato de los datos es el siguiente:

x_1 → B B B ... B B P
x_2 → B B B ... B B P
... .
x_n → B B B ... B B P
x_{n+1} → P P P ... P P P

donde B representa un bit de datos y P un bit de paridad; así, si ocurre un error simple en el bit j-ésimo del residuo i-ésimo, se detecta en los bits de paridad i-ésimo vertical y

j-ésimo horizontal. Posteriormente se ha profundizado en la corrección y detección de errores para conseguir sistemas tolerantes a fallos basados en el Sistema Numérico de Residuos: Etzel y Jenkins [ETZ80] han propuesto definir el RNS sobre un conjunto de N módulos, de los que sólo $L<N$ definen el rango dinámico M a emplear, mientras que el conjunto de los N módulos define un rango $M_T>M$. Para comprobar si existe o no error, se convierten a entero todas las posibles combinaciones de N-1 residuos; si existe error, una de estas combinaciones produce un entero en el rango [0, M-1], y las otras N-1 proporcionan valores en [M, M_T-1]. Sin embargo, si no hay errores, todos producen un valor inferior a M; de este modo, si hay un error, la combinación que produce el entero menor que M indica la localización y el valor del residuo incorrecto. Jenkins y Krozmeier [JEN83] desarrollaron un método similar para el QRNS, el Sistema Numérico de Residuos cuadrático, empleado en aplicaciones en las que se requieren variables complejas, como la FFT; la representación con el RNS convencional de números complejos conduce a la necesidad de dos sumas reales para lograr una suma compleja y de cuatro productos y dos sumas reales para la multiplicación compleja; por el contrario, el QRNS logra que el producto de números complejos necesite sólo dos productos reales.

2.1.3. Esquemas de conversión

Para finalizar esta breve revisión se va a abordar la conversión de valores enteros a su representación en residuos, y viceversa. El primero de estos aspectos se ve condicionado una vez más por el tipo de aplicaciones del RNS; como ya se ha comentado, la complejidad de las soluciones implicando el tratamiento de residuos frente a esquemas aritméticos convencionales sólo se justifica en aplicaciones de alta velocidad, lo que requiere que la adquisición de datos y la conversión de decimal a residuos se ajusten a esta velocidad. Por tanto, si se trata con datos analógicos [RAD98], la conversión A/D debe realizarse en tiempos inferiores a los 10ns, lo que requiere convertidores de tipo flash con un número de bits relativamente pequeño, por lo común entre 6 y 12 bits. Esta corta longitud de palabra permite que la salida del convertidor A/D sirva de dirección para chips de memoria en los que se hayan almacenados los diferentes residuos del valor de entrada para cada uno de los módulos m_i. Para el caso de datos digitales, Szabo y Tanaka

[SZA67] propusieron un método muy simple para generar los residuos a partir de la representación en complemento a dos; si X está representado por el número de t+1 bits:

$$X = -B_t 2^t + \sum_{j=0}^{t-1} B_j \, 2^j$$

donde B_t es el bit de signo, el residuo $x_i = X \bmod m_i$ se genera como:

$$x_i = \left(B_t \left(m_i - 2^t \bmod m_i \right) + \sum_{j=0}^{t-1} B_j \left(2^j \bmod m_i \right) \right) \bmod m_i \qquad \textbf{(2.5)}$$

De este modo, si definimos la función:

$$F_i(j) = \begin{cases} 2^j \bmod m_i & j = 0, \cdots, t-1 \\ m_i - 2^j \bmod m_i & j = t \end{cases} \qquad \textbf{(2.6)}$$

es inmediato que:

$$x_i = \left(\sum_{j=0}^{t} B_j F_i(j) \right) \bmod m_i \qquad \textbf{(2.7)}$$

y los diferentes residuos pueden obtenerse almacenando las funciones F_i en ROM y sumando los contenidos en sumadores módulo m_i.

En lo que se refiere a la conversión desde el RNS a binario, una n-tupla de residuos puede convertirse a un valor entero de diversas maneras; en primer lugar, se puede realizar un producto interno que lleva a conversión por bases mixtas (MRC: Mixed Radix Conversion); este método genera unos coeficientes, de bases mixtas, que se calculan a través de algoritmos recursivos [JUL78, MIL98]. Otro posible método se basa en la aplicación directa del teorema del resto chino, CRT, de forma que:

$$X = \left(\sum_{i=1}^{n} s_i \left(x_i s_i^{-1} \right) \bmod m_i \right) \bmod M \qquad \textbf{(2.8)}$$

donde $s_i = M/m_i$ y s_i^{-1} es el inverso multiplicativo de s_i, de modo que $(s_i^{-1} s_i) \bmod m_i = 1$. El gran inconveniente de este algoritmo es la necesidad de un acumulador módulo M, siendo generalmente M un entero suficientemente grande como para que sea problemático su

tratamiento. Por último, siempre está el recurso, para sistemas de alta velocidad, de almacenar directamente en una ROM la correspondencia entre la representación en residuos y los valores binarios correspondientes.

2.2. ALGUNAS APLICACIONES CLÁSICAS DEL SISTEMA NUMÉRICO DE RESIDUOS

La lectura del apartado anterior puede no dejar muy claras las posibles ventajas del uso del Sistema Numérico de Residuos; incluso, los inconvenientes que plantean la división, los diferentes esquemas de conversión, la búsqueda del compromiso entre velocidad y rango dinámico, etc., parecen hacer del RNS una mera elucubración para poner en juego algunas ideas interesantes o novedosas, pero sin mayor interés práctico. Por tanto, se van a describir algunas de las propuestas clásicas para la aplicación del Sistema Numérico de Residuos, en las que surgen de manera natural éstos y otros problemas relacionados, así como posibles soluciones.

2.2.1. Filtros digitales FIR

Ya en los años 70 se realizaron propuestas para la construcción de filtros digitales de respuesta impulsiva finita, FIR, empleando el sistema de residuos [JEN77]. Este tipo de filtros ha despertado un gran interés dado que representan de manera natural estructuras estables y son mucho menos sensibles a los errores de cuantización que los filtros de tipo recursivo; su principal desventaja es que generalmente requieren un número elevado de términos no nulos para representar adecuadamente la respuesta en frecuencia del filtro, lo que exige el cálculo de un elevado número de sumas y productos durante el período de muestreo y limita seriamente el *data rate* (datos procesados en la unidad de tiempo) del filtro; esto es así ya que el filtro FIR viene descrito por una ecuación del tipo:

$$y(n) = \sum_{k=0}^{N-1} H(k)\, x(n-k) \qquad \textbf{(2.9)}$$

donde $x(n)$ e $y(n)$ son las secuencias de entrada salida del filtro, respectivamente. Cuando el número de coeficientes N es muy grande, técnicas de transformación como la FFT u otras transformadas numéricas proporcionan la implementación más eficiente de estos filtros; sin embargo, las transformadas requieren que los datos se procesen en bloques, lo que hace necesario el uso de buffers e introduce retardos temporales en la salida; lo que es más, los procesadores de FFT introducen errores de cuantización, las transformadas numéricas requieren palabras de gran longitud y ambos esquemas exigen el diseño de un procesador especial para la transformada.

Para valores pequeños de N, el RNS proporciona una implementación directa y más atractiva para el procesamiento en tiempo real, ya que un filtro FIR sólo hace uso de las operaciones que en el RNS se llevan a cabo a gran velocidad, es decir, suma y multiplicación, mientras que no son necesarias aquéllas otras que perjudican grandemente a las prestaciones de este sistema, como detección de signo y división [JEN77]. Esto resulta muy interesante para aplicaciones tales como el procesamiento digital de señales de radar o de imágenes, donde los datos no están cuantizados de forma muy precisa (4-8 bits) y el filtrado debe ser implementado para un elevado data rate en tiempo real; también resulta ventajosa su utilización en sistemas de procesamiento distribuido, en los que un gran número de pequeños procesadores remotos adquieren y filtran datos para su transmisión a un procesador central. Si la velocidad requerida en los procesadores remotos no es muy elevada, pueden implementarse con microprocesadores de 4 bits, de muy bajo coste, que manejen los residuos de forma secuencial en un esquema de tratamiento en serie.

En la implementación del filtro, los coeficientes $H(n)$ que caracterizan el filtro se hallan almacenados en sus representaciones en residuos, mientras que los valores de la entrada en los sucesivos instantes, $x(n)$, pueden ser convertidos a su representación en el RNS por cualquiera de las técnicas mencionadas en el apartado 2.1.3, ya se trate de señales analógicas o de muestras previamente digitalizadas; de este modo, es fácil obtener en paralelo cada uno de los residuos correspondientes a la salida $y(n)$, de forma que sólo queda la conversión de la n-tupla correspondiente a su valor entero.

Otra cuestión es la elección de los módulos, ya que deben proporcionar un rango dinámico adecuado para el filtro; si se elige un gran número de módulos pequeños se consigue una estructura altamente paralela al tiempo que se mantienen a un nivel moderado los requerimientos de memoria para las diversas operaciones y conversiones. Como ejemplo, se puede considerar un filtro paso-banda dual de orden 64 ($H(n)\neq 0$ para $0\leq n\leq 63$), que es representativo de una gran clase de filtros FIR que requieren gran precisión en los coeficientes para evitar una distorsión bastante seria en la respuesta en frecuencia del dispositivo; se puede demostrar que los coeficientes del filtro pueden codificarse con 10 bits manteniendo el error de cuantización 10dB por debajo del valor de rechazo de banda del filtro, respecto del mismo sistema implementado con coeficientes reales, es decir, con una precisión mucho mayor. Si los datos se cuantizan con 8 bits, incluyendo el signo, y el filtro se implementa minimizando el error de redondeo, una cota superior [JEN77] para $|y(n)|$ viene dada por:

$$|y(n)| \leq \max\{|x(n)|\} \sum_{k=0}^{N-1} |H(k)| = 352768 \approx (18.4\ bits)$$

De este modo, el conjunto de módulos {16, 13, 11, 9, 7} proporciona un rango dinámico de unos 17.1 bits, que puede resultar suficiente en vista de la cota anterior, que es extremadamente pesimista. Con estos módulos, es posible usar PROMs de 256×4 bits para implementar las multiplicaciones en módulo 16, 13, 11 y 9, y una PROM de 64×4 bits para la multiplicación en módulo 7; otra PROM de 256×4 bits puede almacenar las tablas de suma de residuos. Con estos datos, Jenkins y Leon [JEN77] realizaron una comparación entre las prestaciones que podía ofrecer para este filtro el uso del RNS y las que se obtienen con otras estructuras convencionales. En concreto, hicieron uso del RNS para la implementación directa de la ecuación (2.9) con PROMs de 30ns de tiempo de acceso y 500mW de consumo y registros de desplazamiento capaces de trabajar a 20MHz, en estructuras con segmentación de cauce; en su trabajo se proponen dos alternativas, que no se van a discutir aquí, una de las cuales es la implementación directa de (2.9), siendo la otra una ligera modificación del algoritmo para reducir el hardware necesario a costa de disminuir la velocidad de funcionamiento.

Estas dos estructuras fueron comparadas con una realización basada en un

multiplicador y un sumador para números reales, en doble precisión, ya que el filtro basado en residuos no provoca cuantización con redondeo; de nuevo, los coeficientes del filtro se almacenan en una memoria; para esta situación se contemplan de nuevo dos alternativas, que no consisten más que en considerar un multiplicador en serie, que realiza un algoritmo de desplazamiento y suma, o un multiplicador paralelo. Por último, la comparación también incluye la implementación "bit slice", propuesta por Peled y Liu en 1974 [PEL74], que agrupa los 64 términos de la convolución finita en grupos de ocho, de forma que cada uno de los bits de las últimas ocho entradas sirve para direccionar una memoria que contiene todas las posibles combinaciones de los coeficientes del filtro correspondientes a dicho grupo; los contenidos de las memorias que son direccionados son desplazados y acumulados adecuadamente y las ocho sumas parciales se pasan a un acumulador-sumador en el que se recoge el resultado final; esta propuesta dio lugar a la técnica conocida como aritmética distribuida, muy extendida hoy en día para la implementación de este tipo de sistemas en dispositivos programables. Esta estructura puede implementarse con chips de PROM o de RAM y, en cualquier caso, el número de datos procesados en la unidad de tiempo que puede conseguirse está limitado por el tiempo de ciclo que forman el acceso a memoria, desplazamiento y suma.

TABLA 2.1

Alternativa	Data rate (kHz)	Circuitos integrados necesarios	Data rate por unidad de coste (kHz/$)
RNS sin ahorro	260.4	126 TTL	2067
RNS con ahorro	173.6	82 TTL	2117
Convencional con multiplicador serie	54.43	77 TTL	680.9
Convencional con multiplicador paralelo	117.48	137 TTL	857.5
Bit slice	744.0	128 ROM	5812
		200 RAM TTL	3720

En la tabla 2.1 se recogen los resultados obtenidos por Jenkins y Leon en su comparación; de la misma se deduce que la implementación del filtro ejemplo basada en

el sistema de residuos supone un compromiso entre coste y velocidad, ya que a pesar de ser tres veces más costosa que la implementación convencional, consigue una velocidad cinco veces superior; de manera similar, es una alternativa más barata que otras de carácter especial, como la aritmética distribuida.

2.2.2. Implementación de la FFT

El algoritmo FFT (Fast Fourier Transform) se emplea para el cálculo de la transformada discreta de Fourier (DFT: Discrete Fourier Transform) en un gran número de aplicaciones, entre las que cabe destacar el análisis espectral para la codificación de voz, cálculo de ecuaciones en diferencias o análisis de filtros [TSE79]. Generalmente, los circuitos que realizan este algoritmo van desde procesadores digitales de propósito general a circuitos integrados VLSI específicos. Una particularidad de la FFT es que no puede ser calculada exactamente y una realización basada en números enteros, con longitud de palabra finita, posee dos fuentes de error de cuantización: los debidos al redondeo y truncamiento de las operaciones aritméticas y los que provoca la inexactitud de los coeficientes, propiciada por la precisión limitada. Cuando la FFT se realiza con un procesador capaz de manejar números reales, los errores de cuantización debidos al convertidor A/D entran en juego. Existen muchas propuestas para implementar la FFT empleando el sistema binario convencional, lo que lleva de forma inevitable a un elevado número de sumadores y multiplicadores binarios, que han de conectarse en estructuras segmentadas si se desean obtener prestaciones elevadas.

Cabe pensar en una implementación de la FFT usando el sistema de residuos; ésta emplearía una estructura muy simple, que incluso podría realizarse completamente en ROM y, que por su naturaleza intrínsecamente paralela, podría realizar operaciones aritméticas exactas con enteros a muy alta velocidad; sin embargo, los errores asociados a coeficientes inexactos y operaciones de escalado seguirían estando presentes.

El problema del escalado es importante ya que en la práctica no se puede realizar un cálculo módulo M, sino que se pretenden aproximar los cálculos que se llevarían a cabo

en un campo infinito de números reales (o complejos, en cualquier caso). El problema deriva del hecho de que, como ya se ha comentado en más de una ocasión, el escalado es una operación complicada en el sistema de residuos, dado que los diferentes dígitos no proporcionan información alguna de manera inmediata sobre la magnitud del número representado; lo que es más, el número resultante del proceso de escalado no es entero, con lo que es preciso habilitar algún mecanismo de estimación o redondeo. Esta propuesta emplea el algoritmo de escalado ideado por Jullien [JUL78], que ya se comentó en el apartado 2.1.2. De todas maneras, un procedimiento viable de escalado para esta aplicación requiere que el factor de escala esté predeterminado (no se permite una normalización dinámica o adaptativa) y que las operaciones de escalado se eviten en lo posible.

Para llevar a cabo la transformada en el RNS se hace necesario determinar la base de la FFT [TSE79] que minimiza el número de multiplicaciones en cascada para un número dado de muestras; es decir, si el número de muestras es $N=r^m$, con r y m enteros positivos (r se denomina base del algoritmo), y puede estructurarse en m etapas; cada etapa incluye N/r transformadas básicas de r puntos. De este modo, la DFT de la secuencia $\{x(n)\}$ de N puntos complejos es:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \qquad k = 0,\ 1, \cdots,\ N-1 \tag{2.10}$$

y:

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k)\, W_N^{-nk} \qquad n = 0,\ 1, \cdots,\ N-1 \tag{2.11}$$

donde $W_N = e^{-j2\pi/N}$ y los $\{X(k)\}$ son complejos; la forma básica de la transformada de r puntos con supresión en el tiempo (*decimation in time*) es la que muestra la expresión (2.12):

$$x_{i+1}(k) = \sum_{n=0}^{r-1} x_i(n) \left(W_N^{\ t}\right)^n W_N^{-nk} \tag{2.12}$$

donde $\{x_i(n)\}$ representan los datos a la entrada de la etapa i-ésima y $(W_N^{\ t})^n$ son los factores

"twiddle" (literalmente, de giro o de vuelta); así, la expresión (2.10) se lleva a cabo en la práctica en dos pasos: primero se multiplican los r puntos de entrada por los factores adecuados y, posteriormente, se calcula la transformada de los r puntos, como se muestra en la figura 2.1. En los algoritmos de supresión en frecuencia, este proceso se realiza a la inversa. Cuando la base de la FFT es 2 ó 4 todos los coeficientes del algoritmo son triviales, de modo que W_r^{nk} vale 0, ±1 ó ±j y r=2 ó r=4. Por tanto, 4 es la mayor base en la que no se requieren multiplicaciones internas en la DFT de r puntos.

Por otra parte, dado que el escalado es una operación no deseable en el RNS, es necesario evitar el desbordamiento eligiendo la base del algoritmo de forma que se minimice el número de multiplicaciones en cascada, a fin de evitar, en lo posible, las operaciones de escalado. La tabla 2.2 muestra el número de multiplicaciones en cascada para diversas bases, dado que cuando el número de muestras N es potencia de 2, la FFT puede realizarse usando bases de la forma 2^k, con k un entero no negativo. En la tabla se asume que las DFTs de 8 y 16 puntos para el cálculo de las FFTs de bases 8 y 16 se realizan según el algoritmo de Winograd [TSE79] para la transformada de Fourier (WFTA: Winograd's Fourier Transform Algorithm); además, la tabla recoge para r=2 y 4 sólo el número de multiplicaciones por los factores $(W_n^t)^n$, mientras que para r=8 y 16 incluye las multiplicaciones por estos factores y las multiplicaciones internas en la DFT

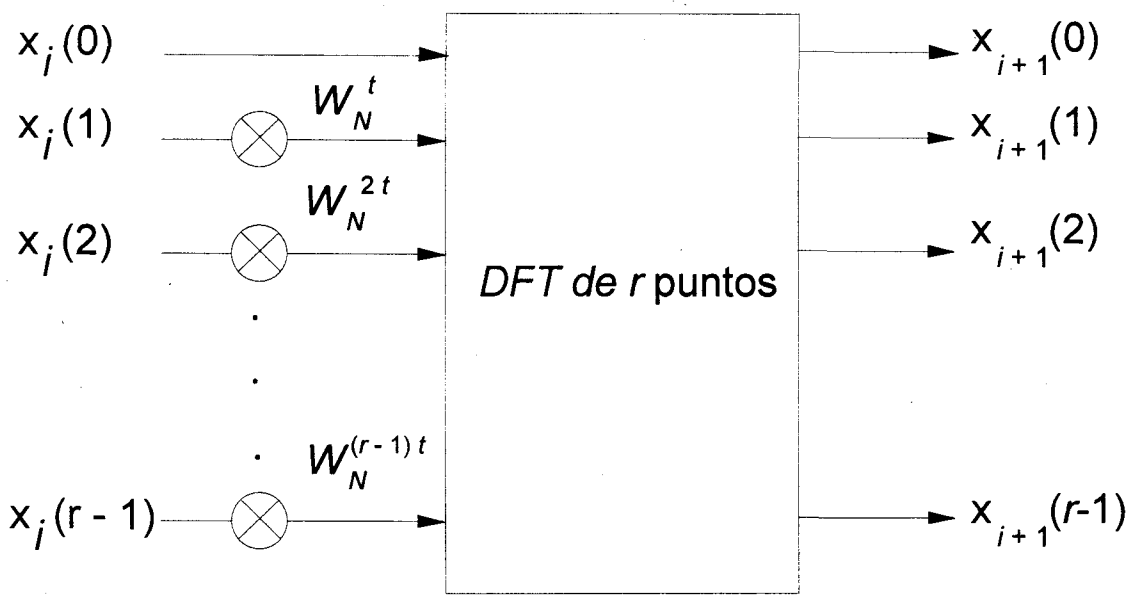


Figura 2.1. Transformada de r puntos.

de r puntos. De la tabla 2.2 se deduce que 4 y 16 son las bases que requieren el menor número de multiplicaciones en cascada, con lo que minimizarán el número de operaciones de escalado; en general, el hardware necesario para la unidad básica de cálculo para una FFT de base 16 es mucho más complejo que para la de base 4. Por tanto, la elección, desde un punto de vista de coste y velocidad, depende de la manera en que se realice el procesador (cascada, secuencial, en paralelo...); sin embargo, la realización en base 16 limita seriamente la viabilidad de un procesador debido al número de muestras N que pueden seleccionarse; es por estos motivos por los que en este caso se considera la base 4 como la elección óptima.

TABLA 2.2

	N=64	N=128	N=256	N=512	N=1024	N=2048	N=4096
r=2	4	5	6	7	8	9	10
r=4	2		3		4		5
r=8	3			5			7
r=16			3				5

De las expresiones (2.10), (2.11) y (2.12) se deduce que el valor cuadrático medio de la transformada es N veces el de la secuencia de entrada, de forma que en cada etapa (recordemos que en el algoritmo en base r hay m etapas, con $N=r^m$) el valor cuadrático medio se aumenta en el factor $N^{1/m}$, que es precisamente r. De este hecho se puede deducir el peor caso teórico en lo que respecta al crecimiento de los números a lo largo del proceso, lo que es necesario para evitar el desbordamiento y realizar el mínimo número posible de operaciones de escalado. De la expresión (2.12) y el crecimiento del valor cuadrático medio se deduce para r=4 que:

$$\frac{\max\left\{\left|Re\left[x_{i+1}(k)\right]\right|,\left|Im\left[x_{i+1}(k)\right]\right|\right\}}{\max\left\{\left|Re\left[x_i(n)\right]\right|,\left|Im\left[x_i(n)\right]\right|\right\}} \le \sum_{n=0}^{3}\left\{\left|\cos\frac{2\pi nt}{N}\right|+\left|\sin\frac{2\pi nt}{N}\right|\right\} \qquad \textbf{(2.13)}$$

Por tanto, se puede observar que el peor caso depende solamente de las magnitudes de los factores $(W_n^t)^n$, que de por sí son independientes de la magnitud de la secuencia de entrada.

La tabla 2.3 muestra las cotas superiores calculadas según la expresión anterior para diversos valores de *N*. De este modo, dichas cotas son independientes del número de muestras, además de que van incrementándose hasta llegar a un valor máximo al añadir más etapas. Esto debe ayudar a determinar los factores de escalado adecuados para evitar que el desbordamiento falsee los resultados.

TABLA 2.3

	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5	Etapa 6
N=64	4	5.027	5.042			
N=256	4	5.027	5.042	5.058		
N=1024	4	5.027	5.042	5.058	5.058	
N=2048	4	5.027	5.042	5.058	5.058	5.058

Más allá de esta frontera teórica que se acaba de calcular, sería deseable encontrar una cota experimental a fin de obtener resultados más realistas; para ello, los autores de la propuesta realizaron un programa que implementa la FFT con base 4 y supresión en el tiempo para 1024 muestras. Este programa fue probado con cinco tipos de señales:

- **1 y 2:** las señales de entrada se componían de números pseudoaleatorios en sus componentes real e imaginaria, no correlacionadas; para el tipo 1 se usó el rango (-1, 1), y para el segundo, (0, 1);

- **3 y 4:** formadas por dos señales senoidales y números pseudoaleatorios, bajo la forma:

$$a(n) + 0.5 \sin\frac{n\pi}{256} + 0.25 \sin\frac{n\pi}{128} + 0.25 \sin\frac{n\pi}{64}$$

  donde $-0.5<a(n)<0.5$ para el tipo 3 y $0<a(n)<1$ para el 4;

- el tipo **5** corresponde a una señal típica de voz.

TABLA 2.4

	Etapa 1	Etapa 2	Etapa 3	Etapa 4	Etapa 5
Tipo 1	3.62	3.89	2.25	1.80	2.15
Tipo 2	3.74	3.95	3.59	3.86	3.84
Tipo 3	2.95	3.08	2.85	3.36	3.76
Tipo 4	3.26	2.67	3.26	3.88	3.95
Tipo 5	2.66	2.95	3.32	4.01	4.00

Los peores casos respecto al crecimiento de los datos procesados para los diferentes tipos de señales se recogen en la tabla 2.4. De estos resultados se deduce que una cota superior razonable para la construcción del procesador de FFT es 4, por debajo de la predicción teórica de peor caso mostrada en la tabla 2.3.

Como ya se ha indicado, el uso de números enteros introduce errores en la FFT, de manera que se hace necesario establecer modelos adecuados para estos errores en orden a minimizar su efecto sobre el circuito final; estos modelos son [TSE79]:

- **Conversión A/D:** los errores más comunes están relacionados con la saturación y la cuantización, pero dado que los niveles de entrada pueden regularse para minimizar los efectos de saturación, sólo han de considerarse los errores de cuantización. Si Q es el tamaño del escalón del conversor (1 en el sistema de números enteros), puede deducirse para el error de cuantización que:

$$\overline{e_Q} = 0 \qquad \overline{e_Q^2} = \frac{1}{12} Q^2 \equiv \frac{1}{12} \tag{2.14}$$

  donde se han despreciado los efectos de la correlación, dado que las fuentes de error siguientes son dominantes.

- **Factores $(W_n^i)^n$:** dado que éstos han de representarse con números enteros, el error asociado ha de determinarse a partir de un factor de conversión p_i, que ha de

introducirse en cada una de las líneas de entrada de la figura 2.1, que pase del valor real a un valor entero, tanto para la parte real como para la imaginaria. En una implementación con ROM, el factor de conversión p_i está predeterminado, y las representaciones enteras de los factores están almacenadas en memoria, de manera que:

$$\overline{e_I} = 0 \qquad \overline{|e_I|^2} = \overline{(Re\{e_I\})^2} + \overline{(Im\{e_I\})^2} = \frac{1}{6} \tag{2.15}$$

- **Escalado:** en la implementación con ROM, la magnitud de los datos en cada etapa debe ser conocida *a priori*, de forma que las consideraciones anteriores sobre el crecimiento de los números permitan elegir un factor de escala capaz de limitar el crecimiento de los datos al valor deseado. Los errores asociados al escalado se deben al redondeo o al truncamiento, dependiendo de la naturaleza del hardware empleado; se puede demostrar que:

$$\overline{e_S} = 0 \qquad \overline{|e_S|^2} = \frac{1}{6} \tag{2.16}$$

Haciendo uso de estos modelos, tras un desarrollo complejo, se llega a la conclusión de que el error asociado a la implementación RNS de un procesador de FFT depende de los factores siguientes:

- σ^2, el valor cuadrático medio de la entrada de la primera etapa;
- m, el número total de etapas;
- p_i, el factor de conversión a entero asociado a los factores "twiddle" en la etapa i-ésima;
- q, el número de operaciones de escalado requeridos en el procesador;
- K, el factor de escalado usado para prevenir el desbordamiento de los datos procesados (se supone constante para todas las operaciones de escalado);
- k_i, el número de etapas entre operaciones de escalado.

Con todo esto, y pensando ya en el diseño, el primer parámetro a determinar es el rango del sistema numérico a utilizar; con éste se puede definir la cota superior permitida

para el crecimiento de los datos en cada etapa, para secuencias de entrada típicas. El resto de los parámetros, p_i, q, K y k_i se determinan *a posteriori*. Una simplificación importante, y muy útil para el diseño, es hacer el número de etapas entre operaciones de escalado constante, que permite, además, un número variable de etapas antes de la primera operación y después de la última.

En cualquier caso, la simulación por ordenador de diferentes esquemas de diseño para el procesador FFT ayuda a ver la relación entre el error asociado al mismo y los diferentes parámetros: es evidente que el error será inversamente proporcional al rango M del sistema de residuos empleado, dado que un mayor rango permite reducir el número de operaciones de escalado necesarias para evitar el desbordamiento y, por tanto, el error que introduce cada una de ellas. Sin embargo, esta tendencia general depende también del esquema de escalado empleado, con lo que es preciso ceñirse a aquéllos para los que el error asociado es mínimo.

TABLA 2.5

$\{m_i\}$	K	α	Realización secuencial	Realización en cascada
32, 31, 29, 13, 3	31×13	0.01	266	1048
32, 31, 27, 23, 7	31×27	0.005	266	1048
31, 19, 17, 13, 11, 7	17×13×11	0.001	372	1416
Velocidad funcionamiento			$8/5\tau$	$8/\tau$

Un problema que surge es la elección de los módulos, dado que, aunque para un determinado nivel de error relativo α es siempre posible seleccionarlos para que su producto sea cercano al valor requerido de M, no pueden proporcionar un valor adecuado para el factor de escalado, que ha de determinarse a partir de los módulos seleccionados. En este caso, puede ser más ventajoso hacer que algunos de los módulos sean menores que lo requerido, a fin de que el conjunto de módulos ofrezca una mayor flexibilidad para obtener el valor deseado del factor de escalado. En la tabla 2.5 se ofrecen algunos valores interesantes para la realización de estas estructuras, en los que se recogen el número de

tablas necesarias para su construcción, con los módulos adecuados, el factor de escalado y el error relativo deseado; para todos ellos, m=5, q=4 y N=1024. Como vemos, el número de tablas no tiene en cuenta el tamaño de las ROMs para los diferentes módulos; en lo que respecta a la velocidad de funcionamiento, τ es el tiempo de acceso a memoria más el tiempo necesario para cargar un registro.

Una posible aplicación, relacionada con lo visto en el apartado anterior, es la construcción de un filtro haciendo uso del procesador FFT construido con el sistema de residuos, siempre que sea aceptable reducir el data rate (datos procesados en la unidad de tiempo) a la mitad. Esta solución, como ya se comentó en el apartado anterior, no es la más adecuada, pero puede ser factible si no se desea realizar una implementación específica; esto es así ya que el filtro ha de implementarse con la estructura de la figura 2.2, haciendo uso de dos procesadores, uno para la FFT y otro para la IFFT (FFT inversa), aunque el mismo hardware puede realizar las dos operaciones en una implementación en serie. Sin embargo, en comparación con los resultados anteriores sobre error relativo y prestaciones, esta estructura es subóptima con relación a la implementación sólo de la FFT, que se puede considerar óptima. Con la estructura de la figura 2.2 se pueden realizar experiencias de error subjetivas, filtrando voz dándole al filtro un gran ancho de banda y

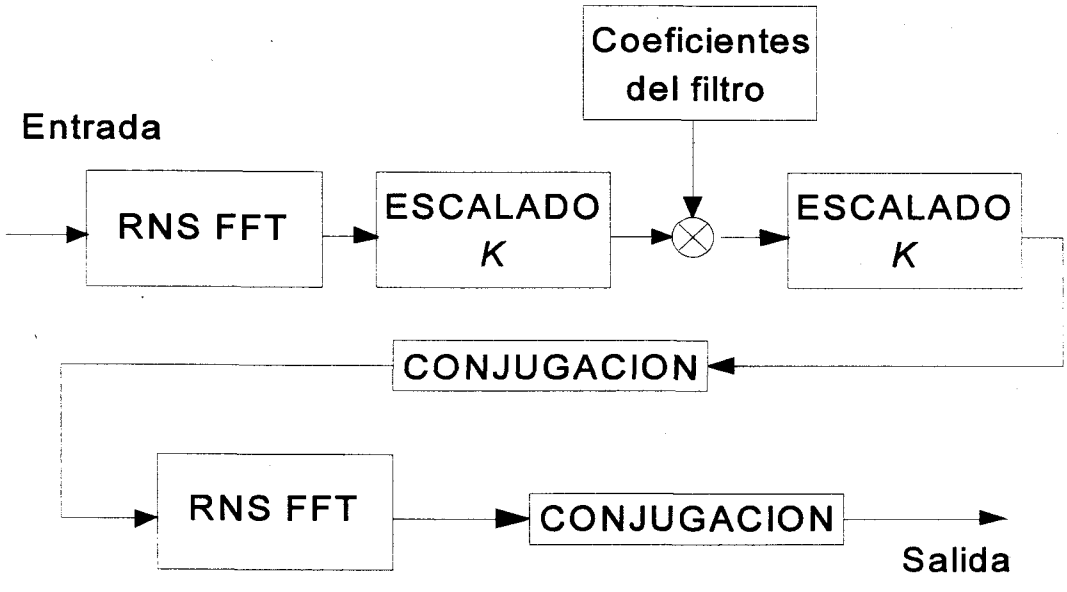


Figura 2.2. Filtro FIR construido con procesadores FFT.

comparando la entrada y la salida. Un resultado curioso es el comprobar que existe una correlación directa entre la calidad subjetiva de la voz de salida y la magnitud teórica del error, como era de esperar, lo que da una idea del buen funcionamiento de la implementación expuesta.

De este modo, esta propuesta [TSE79] muestra que es posible la construcción de un procesador FFT completamente basado en el RNS aplicable a la construcción de filtros de convolución de alta velocidad; lo que es más, el procesador puede implementarse en la práctica haciendo uso sólo de memorias ROM dispuestas como tablas de consulta. Además, el desarrollo de este procesador FFT incluye el tratamiento adecuado para fijar el rango dinámico, la precisión de la conversion A/D y el factor de escalado a fin de minimizar el error cuadrático medio de la salida.

2.2.3. Procesador digital rápido

Se va a tratar ahora una aplicación mucho más específica [HUA81], un procesador que realiza la convolución bidimensional de un filtro lineal con una secuencia de datos, en matrices 5×5, tanto los coeficientes como los datos. Por tanto, para cada salida, la convolución requiere 25 multiplicaciones y 25 sumas:

$$G_{IJ} = \sum_{K=1}^{5} \sum_{K=1}^{5} H_{KL} M_{I-K, J-L} \qquad \textbf{(2.17)}$$

donde G representa la salida, M la entrada y H los coeficientes del filtro. Cuando se minimiza el error cuadrático medio, los coeficientes H dependen únicamente de la estadística del ruido y las características de la señal; si el ruido posee la misma correlación en las dos direcciones y si las características de la señal también coinciden en estas dos direcciones, los coeficientes que minimizan el error cuadrático medio también serán simétricos en ambas direcciones; de este modo, en la matriz 5×5 sólo habrá 6 valores distintos, que se distribuyen de la manera siguiente:

$$\begin{pmatrix} F & E & D & E & F \\ E & C & B & C & E \\ D & B & A & B & D \\ E & C & B & C & E \\ F & E & D & E & F \end{pmatrix}$$

Durante la operación de convolución, la matriz de coeficientes barre de izquierda a derecha las cinco columnas de los datos de entrada (cinco series de datos); a causa de la simetría, la primera y la quinta columnas de los datos se multiplican por los mismos coeficientes y luego se suman antes de la convolución; la segunda y la cuarta se tratan de manera similar. Después de la suma de las columnas, se puede considerar que quedan tres: la primera más la quinta, la segunda más la cuarta y la central, con lo que se puede considerar la matriz de coeficientes como una matriz 3×5:

$$\begin{pmatrix} F & E & D & E & F \\ E & C & B & C & E \\ D & B & A & B & D \end{pmatrix}$$

UNIVERSIDAD DE GRANADA
[illegible]
[illegible] DOCTORADO

Con esto, la convolución requiere quince multiplicaciones y quince sumas, además de las dos adiciones para sumar las dos parejas de columnas. Se puede simplificar más aún el algoritmo sumando términos con el mismo coeficiente del filtro antes de multiplicar por dicho coeficiente; por ejemplo, los cuatro valores de entrada que se multiplican por el coeficiente *E* pueden sumarse antes de la multiplicación, con lo que se transforman cuatro multiplicaciones y una suma en cuatro sumas y una multiplicación, lo que es a todas luces un cambio ventajoso. Haciendo uso de este hecho, la convolución pasa a necesitar sólo seis multiplicaciones (para los seis valores diferentes de los coeficientes) y quince sumas, además de las dos para añadir las columnas.

Esta propuesta se debe al trabajo de Huang, Peterson, Rauch, Teague y Fraser [HUA81], que construyeron un prototipo controlado por un Intel 8086; el hardware necesario para la implementación de la convolución ocupaba 12 tarjetas de circuito impreso, una de ellas para la interconexión con el microprocesador, que además actúa sobre un teclado y un sistema gráfico para la representación de los datos. En esta

propuesta, los datos se representan con 8 bits, y son cargados por el 8086 en dos de las tarjetas; otra de las placas se encarga de sumar las cinco series de datos de entrada para formar las tres que son necesarias para el cálculo; en esta misma tarjeta existe un codificador que pasa los datos de la representación usual binaria al Sistema Numérico de Residuos con un rango dinámico de 18 bits. En este punto es necesario elegir los módulos con los que trabajar, que en este caso son {16, 15, 13, 11, 7}, que se ajustan al rango dinámico mencionado; existe una tarjeta para cada uno de estos módulos, que son las encargadas de realizar las operaciones correspondientes a cada uno de ellos. Dos de las tarjetas restantes convierten los datos de salida del RNS a su representación convencional, que se almacenan en un buffer de salida en la última de las tarjetas. Si para este sistema se usan PROMs de 256×4 bits en tecnología ECL de 25ns de tiempo de acceso, el sistema es capaz de realizar 30 millones de convoluciones por segundo.

Respecto a la conversión de binario al sistema de residuos, ya se ha dicho que cuatro de las columnas se suman dos a dos, siendo las entradas de 8 bits; por tanto, es posible llevar a cabo esta suma con dos sumadores de 4 bits. Cada número de 8 bits ha de generar 5 residuos, lo que se hace a través de tablas; sin embargo, no es necesario emplear cinco memorias, una para cada módulo, ya que el residuo para módulo 16 lo constituyen directamente los 4 bits menos significativos de la suma (o de la entrada, si se trata de la columna central que no se suma a ninguna otra). Por tanto, para las sumas se emplean dos tablas ROM de 256×4 bits para cada módulo (excepto para 16, como ya hemos dicho), de forma que el posible bit de desbordamiento decide qué ROM se emplea de cada pareja, y los 8 bits de la suma sirven de dirección para los módulos seleccionados; la entrada central sólo requiere una ROM de 256×4 bits para cada uno de los residuos a generar, en los que el propio valor actúa como dirección. Es preciso asegurar que para cada una de las series de datos (tanto para la central como para las que son sumas) el retardo es el mismo, con lo que deberán incluirse en el tratamiento de la serie central los retardos necesarios para compensar la ausencia de suma; esto es así para garantizar una correcta sincronización que permita dotar al sistema de segmentación de cauce.

En lo que se refiere al módulo que implementa la operación del filtro para cada uno de los residuos, consiste en tres líneas de retardo seguidas por una red sumadora en la que

se combinan todos los términos de la señal que han de multiplicarse por el mismo coeficiente. Cada sumador en esta red está construido con una PROM de 256×4 bits, de forma que las dos entradas se usan como dirección a la memoria, que contiene el resultado adecuado para cada combinación de entradas. Un conjunto de 6 ficheros de registros 16×4, uno por cada valor posible de los coeficientes, almacenan los 16 patrones de coeficientes; éstos se multiplican por los resultados de la red sumadora usando PROMs de 256×4 bits en las que se almacenan las tablas de multiplicación para cada uno de los diferentes módulos; por último, las 6 salidas se suman en una nueva red sumadora de PROMs. El hecho de que cada fichero de registros contenga 16 registros de 4 bits permite que el sistema propuesto sea capaz de implementar múltiples filtros, de forma que puede realizar varias combinaciones; de este modo, si CKF es el reloj que selecciona los diferentes registros para cada módulo y el decodificador y buffer de salida, y CKS es el reloj que controla el buffer de entrada y el codificador (de binario al sistema de residuos), CKS ha de ser K veces más lenta que CKF para obtener la salida correspondiente a K conjuntos de coeficientes diferentes.

Por último, dos placas implementan el algoritmo de conversión del RNS al sistema binario convencional, que está basado en el teorema del resto chino, tal y como se indicaba en el apartado 2.1.3. En la primera placa, las salidas de los filtros correspondientes a cada uno de los módulos se combinan en parejas y se multiplican por el coeficiente adecuado según el CRT; estas operaciones se implementan con PROMs; en la segunda de las placas, las salidas de la tarjeta anterior se suman módulo 240240 (que es el rango dinámico que permite el conjunto de módulos elegido) y se traducen a complemento a dos con 18 bits.

Véase más detenidamente la forma de implementar el algoritmo del CRT; en la propuesta original, aunque el M de la ecuación (2.8) es 240240, el sumador en módulo M se remplaza por un sumador y un comparador en módulo $262244=2^{18}$, que son más fáciles de implementar y más rápidos. El resultado se obtiene en complemento a dos, lo que difiere de otros métodos, que requieren una conversión desde un entero positivo a complemento a dos. En este caso, en lugar de la expresión (2.8) se emplea una modificación, de forma que el número X correspondiente a la tupla $[x_{16}, x_{15}, x_{13}, x_{11}, x_7]$ es:

$$X = \left[\left(A_{16}x_{16} + A_7x_7\right) \bmod M + \left(A_{13}x_{13} + A_{11}x_{11}\right) \bmod M + A_{15}x_{15}\right] \bmod M$$

donde:

$$M=240240 \qquad A_{16}=105105$$
$$A_7=205920 \qquad A_{13}=36960$$
$$A_{11}=196560 \qquad A_{15}=176176$$

La cantidad entre corchetes se sitúa entre 0 y $3M$, y es claro que M=240240 y $2^{17}<M<2^{18}$. Si:

$$Q = Q_1 + Q_2 + Q_3 =$$
$$= \left(A_{16}x_{16} + A_7x_7\right) \bmod M + \left(A_{13}x_{13} + A_{11}x_{11}\right) \bmod M + \left(\left(A_{15}x_{15}\right) \bmod M + 2^{18} - M\right)$$

entonces $2^{18} - M \le Q < 3M + 2^{18} - M$. En la primera etapa del decodificador, los residuos se combinan en parejas usando ROMs para formar Q_1, Q_2 y Q_3; tras esto, Q_1, Q_2 y Q_3 se suman usando dos sumadores de 20 bits, ya que 20 bits son suficientes para representar Q. Los dos bits más significativos de Q, q_{19} y q_{18}, se emplean para determinar las posibles correcciones:

1) si $\boldsymbol{q_{19}=0}$ y $\boldsymbol{q_{18}=0}$, Q se compara con $2^{18} - M/2$, y si es mayor o igual que ésta cantidad no se requiere corrección alguna; en caso contrario, M se suma módulo 2^{18} a Q;

2) si $\boldsymbol{q_{19}=0}$ y $\boldsymbol{q_{18}=1}$, el punto de comparación es $M/2$. Si Q es mayor o igual en módulo 2^{18} que ésta cantidad, se suma a Q la cantidad $(2^{18} - M)$ en módulo 2^{18}; en caso contrario, no es necesaria corrección alguna;

3) si $\boldsymbol{q_{19}=1}$ y $\boldsymbol{q_{18}=0}$, la comparación se realiza con $3M/2 - 2^{18}$, y si Q es mayor o igual en módulo 2^{18} que ésta cantidad, se suma $(-2M)$ en módulo 2^{18} a Q; en caso contrario, no es necesaria corrección alguna;

4) $\boldsymbol{q_{19}=1}$ y $\boldsymbol{q_{18}=1}$ no es posible, ya que $Q<3M+2^{18}-M<2^{19}+2^{18}$.

Como se ve, todas las comparaciones y adiciones se realizan en módulo 2^{18}, lo que resulta muy interesante, ya que el sumador no es más que un sumador convencional de 18 bits en el que se descarta el acarreo, y el comparador es uno binario de 18 bits; además, el Q resultante es un número binario de 18 bits convenientemente formateado en complemento a dos. El esquema se recoge en la figura 2.3, en la que COM representa el generador de la constante de comparación y COR el generador del factor de corrección. En la primera placa se lleva a cabo la generación de Q_1, Q_2 y Q_3, necesitándose $2^8 \times 20$ bits de ROM; el resto del diagrama de la figura 2.3 se implementa en la segunda de las placas.

A este esquema se le pueden añadir algunas funciones interesantes, como la detección de desbordamiento, de forma que si se observa que la secuencia de salida se va acercando a valores demasiado grandes, se puedan escalar los diferentes coeficientes. También puede reducirse el hardware necesario usando chips de ROM con salidas con biestables incorporados, ideal para la implementación de la segmentación de cauce. Obviamente, el esquema aquí propuesto es sólo un prototipo, y la integración de todo el procesador en un circuito LSI o VLSI es factible sin afectar a las prestaciones; en cualquier caso, este esquema funciona sin errores realizando 20 millones de operaciones por segundo, lo que da una idea de las razones para el uso del sistema de residuos: una alta

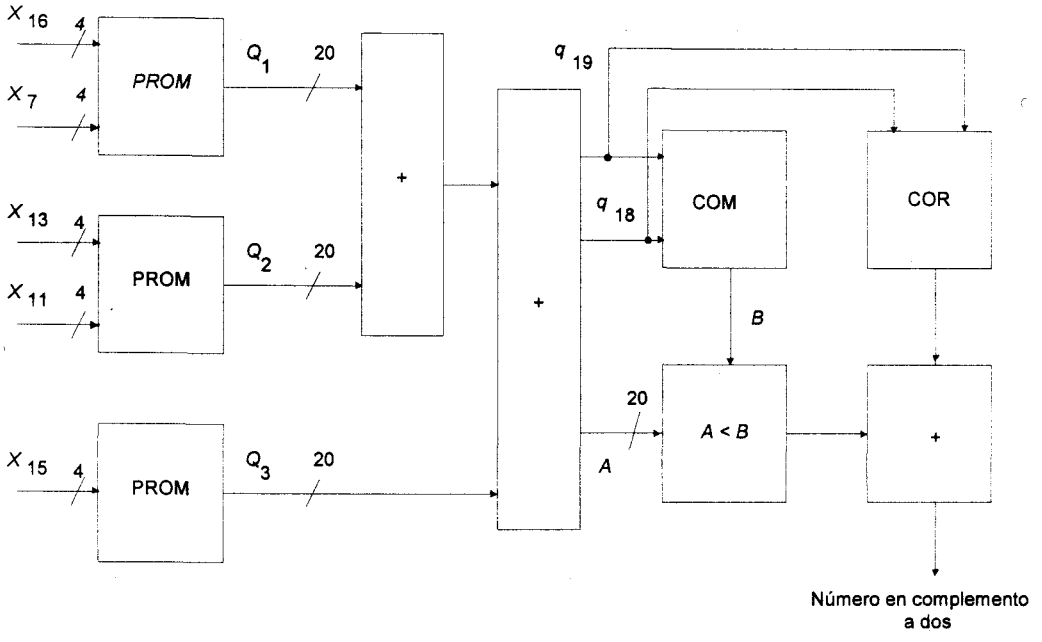


Figura 2.3. Implementación del CRT.

velocidad de funcionamiento con un hardware muy sencillo, básicamente formado por ROM, lo que es cada día más barato a medida que mejora la tecnología.

2.2.4. Síntesis directa digital de frecuencias

Para finalizar con esta sección, se presenta una aplicación nueva del RNS [CHR95], como es la síntesis directa digital de frecuencias. Recientemente, la síntesis digital directa se ha convertido en un método muy empleado para la síntesis de frecuencias, debido a aplicaciones muy exigentes, tales como el radar Doppler, la espectroscopia e imágenes por resonancia magnética o el diseño de equipos para test de sistemas. Las ventajas de este método frente a los sintetizadores basados en PLL (Phase-Locked Loop) se cifran en su gran resolución, el cambio de frecuencias rápido y continuo, la gran linealidad y el buen comportamiento ante los cambios de temperatura y el envejecimiento.

La mayoría de métodos para síntesis digital de frecuencias se basan en una tabla de consulta que almacena muestras digitales de una señal senoidal, a la que se accede periódicamente; posteriormente, las muestras se convierten con un conversor D/A apropiado. Las direcciones para la ROM con la señal senoidal se generan con un acumulador de fase. Supongamos que muestras equiespaciadas del periodo de la señal coseno se almacenan en una ROM de 2^L palabras; de este modo, en la dirección m se halla almacenado el valor:

$$S_m = \cos\left(\frac{2\pi}{2^L} m\right) \qquad (0 \le m \le 2^L - 1) \tag{2.18}$$

Las muestras almacenadas en las posiciones cuya dirección es un múltiplo lk (l=0, 1,...) de la palabra de selección de frecuencia k se convierten mediante un conversor D/A; estas direcciones lk se generan con el llamado acumulador de fase, que no es más que un sumador módulo 2^L dispuesto como acumulador y con k (expresada con L bits) como entrada. Si $f_c = 1/\Delta t$ es la frecuencia a la que se accede a la tabla, la tensión que genera el conversor D/A viene dada por:

$$V_o(l\Delta t) = \cos\left(2\pi \frac{f_c}{2^L} k l \Delta t\right) \tag{2.19}$$

Por tanto, la frecuencia de la señal generada es:

$$f = \frac{f_c}{2^L} k \tag{2.20}$$

Los tres parámetros que definen las prestaciones del sintetizador son las resoluciones en frecuencia, fase y amplitud; obviamente, la resolución en frecuencia es $f_c/2^L$, y suele especificarse a través de L, en bits. Generalmente, las implementaciones prácticas de este método buscan reducir el tamaño de la tabla de coseno, de forma que los L bits de salida del acumulador de fase se truncan hasta W; más aún, los dos bits más significativos de la dirección pueden usarse para almacenar sólo un cuarto del periodo del coseno; así, el bit más significativo se lleva hasta la salida de la tabla para cambiar el signo de la muestra generada, mientas que el otro bit se usa para invertir o no los W-2 bits menos significativos del resto de la dirección.

Chren [CHR95] propone la implementación del sintetizador digital de frecuencias basado en el RNS mostrada en la figura 2.4; para ello se requiere un RNS definido por $\{m_1, m_2, ..., m_n\}$ con $m_i=2^{p+2}$, siendo p un entero positivo. Por otra parte, los elementos

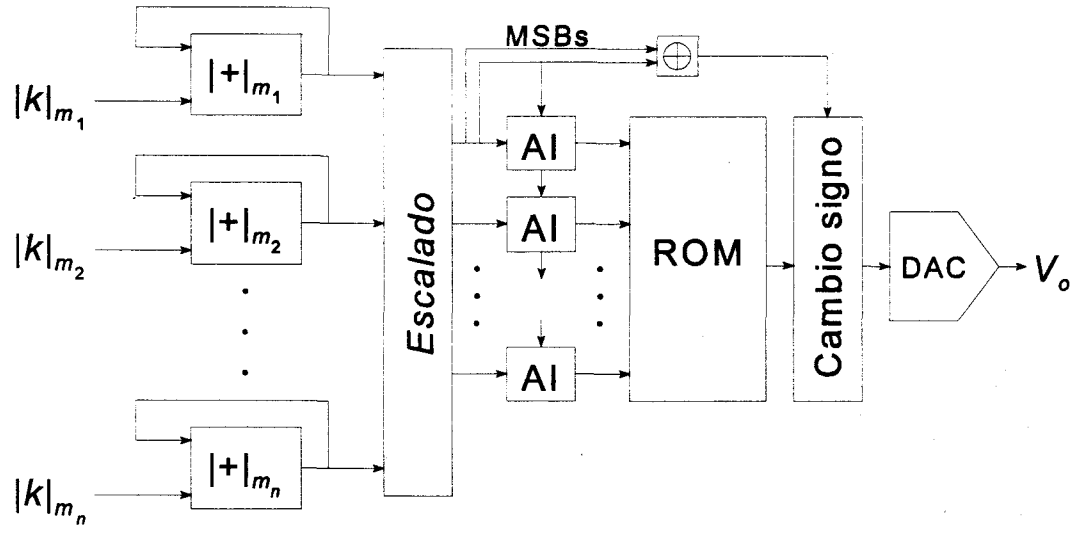


Figura 2.4. Sintetizador digital de frecuencias basado en el RNS.

señalados como AI en la figura 2.4 calculan el inverso aditivo de la correspondiente entrada y tienen como objeto el poder almacenar en la tabla sólo un cuarto del periodo de la señal coseno. Por otra parte, la constante de escalado es $m_{r+1}\times m_{r+2}\times\cdots\times m_n$ ($r>1$) y esta operación se calcula con el algoritmo propuesto por Jullien [JUL78]. De este modo, si se define:

$$2^{p+2}s = m_1\times m_2\times\cdots\times m_r$$

las muestras a almacenar en la tabla están dadas por:

$$\cos\left(\frac{2\pi}{2^{p+2}s}\Phi\right) \tag{2.21}$$

donde Φ es la salida del circuito que calcula el escalado. Si se tiene en cuenta la conversión de RNS a binario a través del CRT (v. apartado 2.1.3) con la ecuación (2.8), la ecuación (2.21) puede calcularse como:

$$\cos\left(\frac{2\pi}{2^{p+2}s}\Phi\right) = \cos\left(\pi s_1^{-1}a_{p+1} + \frac{\pi}{2}s_1^{-1}a_p + \frac{2\pi s_1^{-1}}{m_1}|\Phi|_{2^p} + \frac{2\pi s_2^{-1}}{m_2}|\Phi|_{m_2} + \cdots + \frac{2\pi s_r^{-1}}{m_r}|\Phi|_{m_r}\right) \tag{2.22}$$

donde a_{p+1} y a_p son los bits más significativos de $|\Phi|_{2^{p+2}}$. Con la aproximación de cuarto de onda, sólo es necesario calcular la ecuación (2.22) con los r términos de la derecha, mientras que los bits a_{p+1} y a_p se usan para activar las unidades AI y la unidad de cambio de signo; mientras tanto, la ecuación (2.22) con sólo r términos es la que se calcula a través de la tabla.

La estructura de la figura 2.4 realiza una conversión de RNS a binario, para posteriormente realizar una conversión D/A desde la representación binaria en complemento a dos. Este esquema puede modificarse para realizar una conversión directa del RNS a analógico; además, se pueden emplear r tablas, una para cada uno de los módulos restantes tras el escalado, y combinar las salidas de estas tablas para obtener el

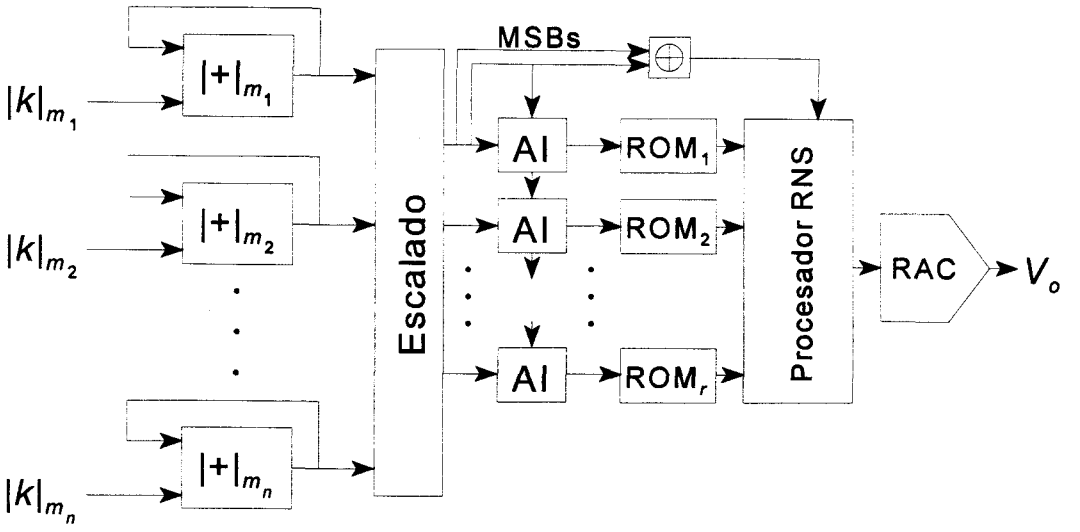


Figura 2.5. Sintetizador digital de frecuencias basado en el RNS modificado.

valor que ha de convertirse en analógico. De este modo, la estructura de la figura 2.5 se traduce en una importante reducción de área con respecto a la figura 2.4, principalmente por el ahorro de la memoria requerida. La tabla *i*-ésima debe almacenar los valores:

$$\cos\left(\frac{2\pi s_i^{-1}}{m_i}|\Phi|_{m_i}\right) \qquad \sin\left(\frac{2\pi s_i^{-1}}{m_i}|\Phi|_{m_i}\right) \tag{2.24}$$

Por otra parte, el procesador RNS en la figura 2.5 calcula en primer lugar el seno y coseno de la suma de las entradas, para luego escalarlos en diferentes maneras y calcular la entrada adecuada para el conversor RNS-analógico (RAC en la figura).

Las ventajas del uso del RNS para la síntesis digital de frecuencias pueden resumirse en:

- El sintetizador de la figura 2.4 reduce el tiempo necesario para el cambio de la frecuencia de salida (un parámetro muy importante) de manera significativa respecto de las implementaciones binarias; esto, lógicamente, se acentúa al crecer la resolución en frecuencia y la reducción supera el 30% para resoluciones en frecuencia superiores a 40 bits. Este efecto se debe a la reducción de la propagación de acarreos en el acumulador de fase, lo que además se traduce en un menor consumo de potencia.

- La alternativa de la figura 2.5 implica una reducción muy importante del área ocupada por el circuito frente a las implementaciones binarias; en concreto, la reducción de área es del 40% para resolución en fase de 13 o más bits, y aumenta conforme crece la resolución en fase y frecuencia, lo que lo hace ideal para diseños de bajo ruido.

De este modo, el RNS se convierte una vez más en una alternativa a los sistemas convencionales binarios, tanto por velocidad como consumo de potencia y área.

2.3. CONCLUSIÓN

En este capítulo se han tratado de introducir los conceptos fundamentales de la aritmética de residuos y el Sistema Numérico de Residuos, al mismo tiempo que se han presentado elementos motivadores del trabajo. De este modo, se habrá podido comprobar que el RNS tiene su campo de aplicación básico en el procesamiento digital de señales, con transformadas, filtrado, etc. Esto se debe al elevado número de sumas y multiplicaciones que suponen este tipo de aplicaciones, ya que éstas son las operaciones en las que el Sistema Numérico de Residuos puede desplegar todas sus ventajas. De este modo, se han presentado algunas aplicaciones clásicas del RNS, mientras que en los capítulos sucesivos se introducirán estructuras aritméticas del RNS y se mostrarán las aplicaciones desarrolladas en este trabajo.

CAPÍTULO III

MÓDULOS ARITMÉTICOS PARA RNS

En este tercer capítulo se introducirán los fundamentos para la realización de las operaciones básicas en el RNS: suma, producto y escalado. Se prestará especial atención a aquellos esquemas que servirán de base a las diversas contribuciones de esta memoria, así como otros que presentan un especial interés por su sencillez, versatilidad, etc. También se presentará un algoritmo típico de división para el RNS, mostrando la dificultad que conlleva esta operación.

3.1. SUMADORES EN ARITMÉTICA DE RESIDUOS

La suma es una de las operaciones básicas en cualquier sistema digital y es una de las que encuentra enormes ventajas en el RNS. En esta sección se presentarán las arquitecturas básicas para sumadores en módulo, con especial énfasis en aquéllas que han sido usadas en todo o en parte en el desarrollo de esta memoria. El primer diseño que se va a exponer es el adecuado para el caso en el que el módulo del sumador es pequeño, incluyendo la posibilidad de realizar la resta con el mismo circuito sin ningún tipo de penalización. También se propondrá un diseño para módulos mayores, lo que puede resultar interesante para aquéllas aplicaciones en las que no sea posible tratar los datos con un número elevado de módulos diferentes.

3.1.1. Sumadores convencionales módulo *m*

Una gran variedad de diseños para el RNS ha surgido a lo largo de los últimos 20 años, yendo desde implementaciones SSI o MSI a las construcciones ideadas para su integración VLSI en los últimos tiempos. Las diferentes estrategias de diseño aplicadas pueden clasificarse en:

- implementación lógica directa empleando puertas;
- uso de sumadores binarios;
- implementación con tablas (métodos "table look-up");
- métodos híbridos.

El primer tipo de diseño fue bastante popular a comienzos de los 70 [BAN74], dado que el uso de puertas integradas en SSI o MSI era muy común, mientras que las memorias eran caras y no excesivamente rápidas. El principal objetivo de esta aproximación es representar los números $\{0, 1,..., m\text{-}1\}$ con una red de puertas lógicas para posteriormente implementar algún tipo de mapeo que lleve a cabo la suma.

A principio de los años 80 surgieron algunas propuestas que se ajustan a lo que se han llamado métodos híbridos [SOD86], orientados a la construcción basada en unidades básicas MSI o LSI, tales como un sumador completo o una PLA.

Posteriormente, Bayoumi, Jullien y Miller [BAY87a] estudiaron el diseño y la implementación de sumadores RNS a nivel de circuito; este estudio daba expresiones analíticas para estimar el área y la complejidad.

La aproximación basada en sumadores binarios requiere dos sumadores con acarreo anticipado y un multiplexor, tal y como muestra la figura 3.1, y lo único que se pretende es implementar directamente la suma en el sistema de residuos:

$$(X+Y) \bmod m = \begin{cases} X+Y & \text{si } X+Y<m \\ X+Y-m & \text{si } X+Y\geq m \end{cases} \qquad \textbf{(3.1)}$$

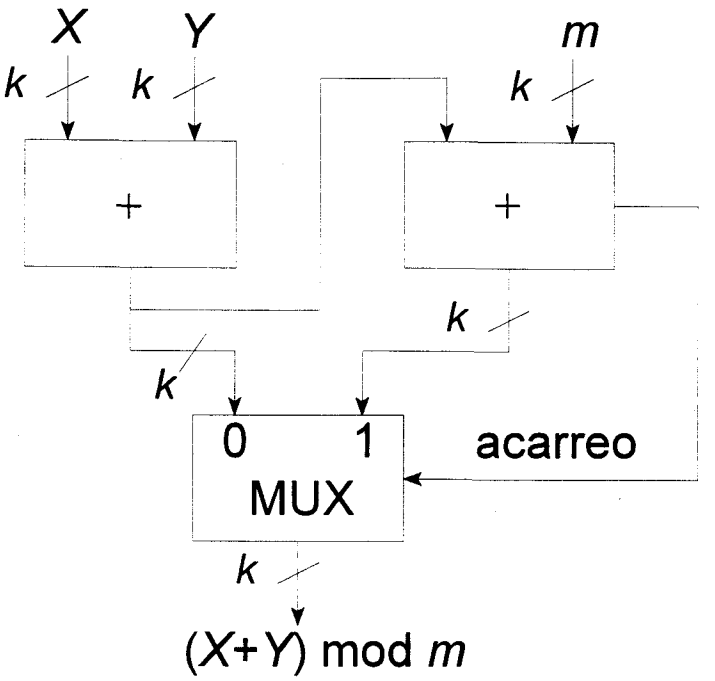


Figura 3.1. Sumador RNS basado en sumadores binarios.

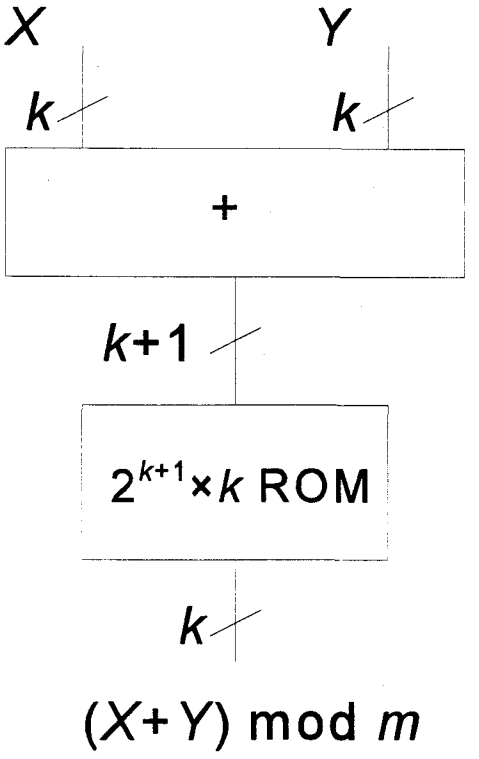


Figura 3.2. Solución híbrida.

con la ventaja de que si $m=2^k$, con k entero positivo, el residuo y la representación binaria son iguales, y un sumador binario de k bits proporciona la suma módulo m sin más que despreciar el acarreo. En la figura 3.1 se supone que $k=\lceil \log_2 (m-1) \rceil$, es decir, que son suficientes k bits para representar todos los números comprendidos entre 0 y m-1.

Por su parte, la alternativa basada en el uso de tablas, asumiendo de nuevo que son suficientes k bits para operar en módulo m, sólo requiere una ROM de m^2k bits, a la que se le aplican como direcciones los dos sumandos ($2k$ bits de direcciones, lo que implica que el número de palabras de k bits de la memoria es mayor que m^2) para direccionar una palabra de k bits en la que previamente se ha almacenado el valor de la suma de los dos operandos en módulo m. Es evidente que, salvo que se utilicen esquemas especiales para esta ROM, se desperdician $2^{2k}-m^2$ palabras, ya que $m-1<2^k$. Esto supone que es necesario realizar una cuidadosa optimización del valor de m a elegir, así como de las memorias a emplear.

Una alternativa intermedia es el llamado sumador híbrido, que se muestra en la figura 3.2, en la que se utilizan un sumador binario y una tabla ROM. El sumador calcula la suma ordinaria de las entradas, que actúa como dirección de la ROM; la tabla se emplea para corregir dicha suma en el caso en que el resultado sea mayor que m. De este modo, la dirección que se emplea es de k+1 bits, con una salida de k bits para la suma módulo m, por lo que el tamaño de la ROM a emplear es $2^{k+1}\times k$ bits y hay un ahorro en la cantidad de memoria con respecto a la alternativa anterior.

Existen estudios [BAY87a] sobre la estructura más adecuada para implementar los sumadores en RNS según el valor del módulo empleado:

- si $m=2^k$ y $k\leq 4$, las tablas ROM son la mejor solución en cuanto a coste;
- si $m=2^k$-1 y $k\geq 5$, las tablas ROM proporcionan los sumadores más rápidos;
- si $m < 2^k$-1 (que es lo más usual) la alternativa basada en sumadores binarios proporciona la mejor solución en lo que se refiere a área y velocidad, suponiendo $k\geq 6$.

3.1.2. Método de rotación-selección

Este método fue propuesto por Banerji [BAN74] y está basado en el hecho de que el conjunto $\{0, 1,..., m\text{-}1\}$ es un grupo cíclico respecto a la operación suma en módulo m, donde m se puede representar con k bits.

El método parte de los elementos del conjunto almacenados en un registro de desplazamiento (cada uno de los números en una "posición" del registro): dados dos enteros X e Y en el rango $[0, m\text{-}1]$, su suma $(X+Y)$ mod m puede obtenerse rotando el registro a la izquierda X veces y seleccionando posteriormente el elemento en la posición Y-ésima del registro. Después de la rotación, el orden de los elementos en el registro, de izquierda a derecha es:

$$\{X, X+1,..., m\text{-}2, m\text{-}1, 0, 1,..., X\text{-}1\}$$

con lo que el Y-ésimo elemento corresponde a $(X+Y)$ mod m, como se deduce rápidamente. Para implementar este mecanismo Banerji propuso una red de km^2 puertas NAND de 2 entradas para realizar la lógica de rotación y una línea de km puertas para la lógica de selección (la lógica de rotación puede realizarse con otras estructuras, que reducen el número de puertas empleadas pero no son adecuadas para sumadores rápidos).

A fin de reducir el hardware, se puede modificar el esquema anterior sin más que, dadas las entradas X e Y, generar una cadena de m bits con un uno en la posición X-ésima (de derecha a izquierda) y cero en el resto; dado que el número de ceros que sigue al uno es X, la cadena de bits representa la suma de X con 0. Si esta cadena se rota a la izquierda cíclicamente Y veces, la posición del uno será $(X+Y)$ mod m. La cadena puede usarse como señal de selección para una ROM de m palabras que contenga el conjunto de enteros representables en módulo m en orden decreciente $\{m\text{-}1, m\text{-}2,..., 1, 0\}$. Puesto que la posición del uno es $(X+Y)$ mod m, la ROM dará como salida el valor correcto de dicha suma.

Un ejemplo del algoritmo con m=8, X=5 e Y=6 se ilustra con la figura 3.3a; de este modo, la cadena de 8 bits es 00100000, que al ser rotada 6 veces produce 00001000; como se puede observar, la posición del uno corresponde ahora a 3, siendo precisamente (5+6) mod 8 = 3. Esta operación puede llevarse a cabo con el circuito propuesto en la figura 3.3b.

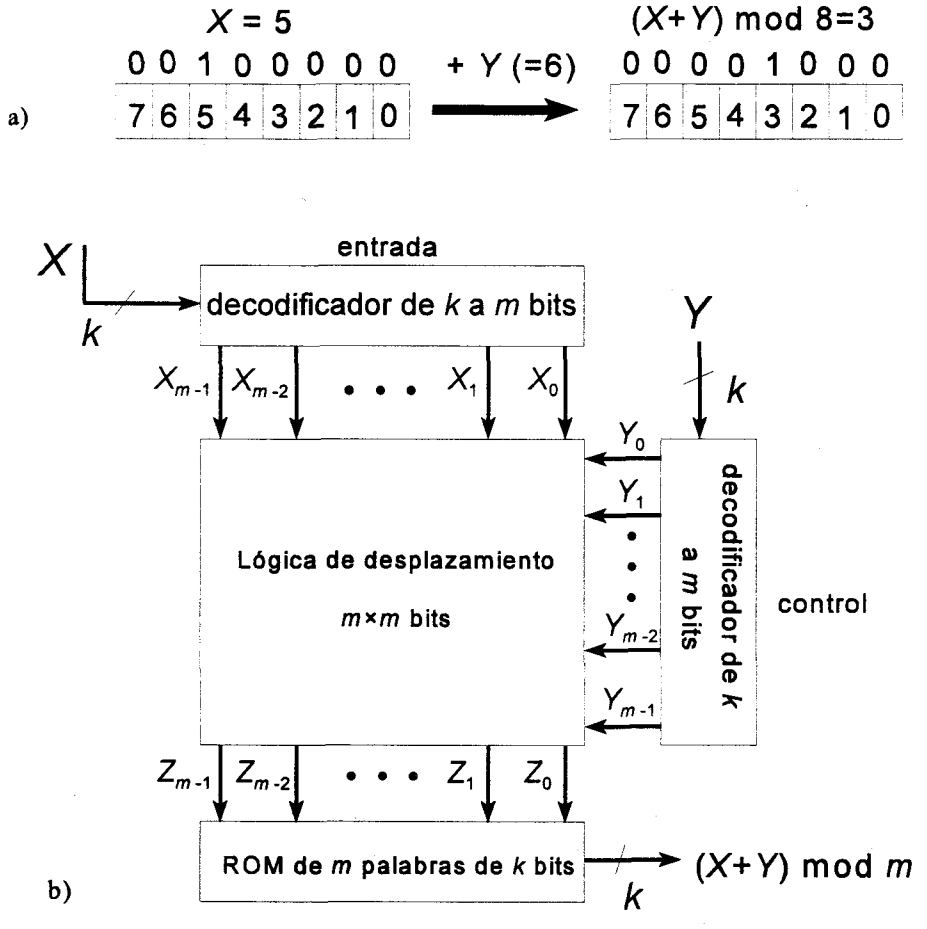


Figura 3.3. a) Ejemplo del proceso de suma (método de rotación-selección).
b) Estructura del sumador módulo m.

El decodificador de k a m bits para la entrada X genera la cadena binaria de m bits $\{X_{m-1} \cdots X_0\}$; el control de la lógica de rotación calcula $\{Y_{m-1} \cdots Y_0\}$, que indica en qué medida ha de rotarse la cadena anterior, tal y como ocurría en el último ejemplo; la cadena binaria que resulta, $\{Z_{m-1} \cdots Z_0\}$, se aplica a una memoria de m palabras de k bits, en la que se ha almacenado la representación de todos los enteros módulo m en orden decreciente.

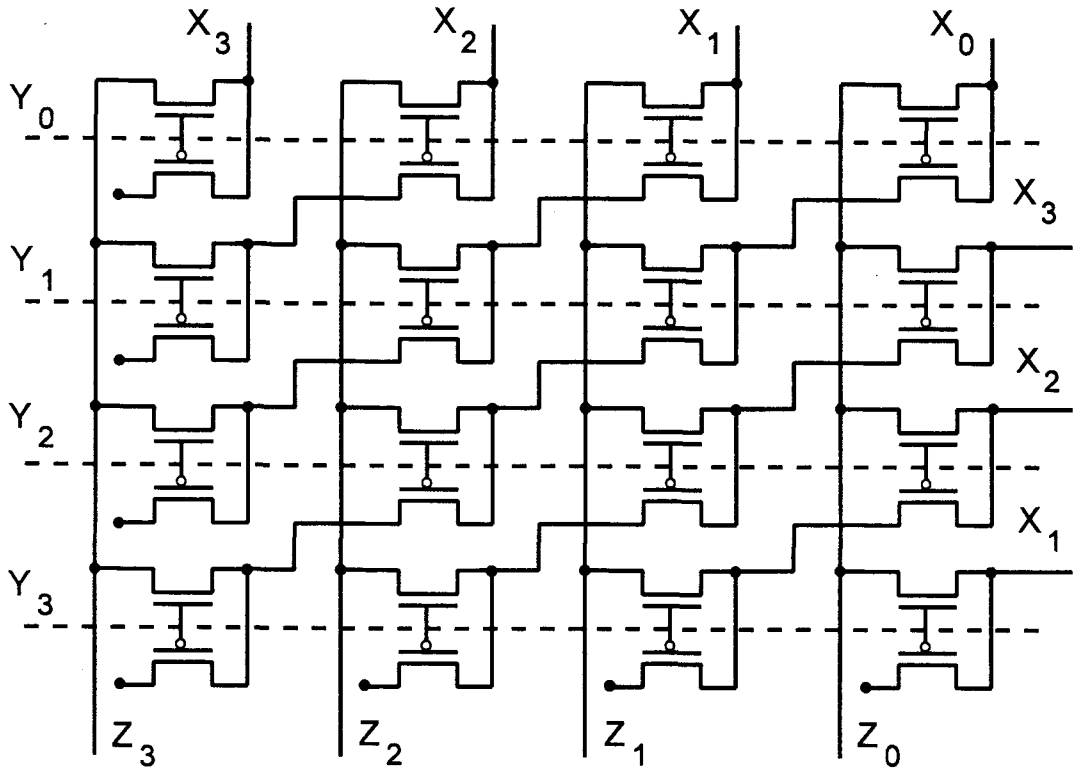


Figura 3.4. Desplazador de barril modificado 4×4.

La lógica de rotación puede realizarse de diversas maneras; por ejemplo, Banerji [BAN74] propone el uso de m^2 puertas NAND, como ya se ha citado, que llevan a una reducción en un factor k de la complejidad en tamaño del circuito. Actualmente, dado el estado de la tecnología, si esto hubiera de llevarse a la práctica para un valor no excesivamente grande de m, lo ideal sería una implementación VLSI con transistores, en lugar de puertas o entidades más complejas. En concreto, es bien conocido el desplazador de barril [UYE92], que consiste en una matriz de celdas muy básicas (prácticamente un transistor por celda) conectadas adecuadamente; la figura 3.4 recoge la estructura ligeramente modificada de uno de estos desplazadores con una matriz de 4×4 celdas formadas por una pareja de transistores NMOS y PMOS. La estructura del desplazador es muy regular y fácilmente realizable en un área muy reducida, en su mayoría dedicada a las interconexiones, y como hemos dicho, ideal para una implementación VLSI; además, este esquema puede extenderse para un mayor número de bits conservando la estructura. Por

tanto, es necesario modificar la celda básica del desplazador clásico de barril (un transistor NMOS) para incluir la rotación (y no sólo el desplazamiento), de forma que cada fila de la matriz con $Y_j=0$ realiza una rotación a la izquierda a través de los transistores PMOS, hasta la fila en que $Y_i=1$, que conecta a través de los NMOS los datos rotados a las líneas de salida. De este modo, si $Y_i=1$ e $Y_j=0$ ($i\neq j$), la estructura de la figura 3.4 rota la cadena de entrada i posiciones hacia la izquierda, que es precisamente la funcionalidad requerida para realizar la suma. La estructura que aquí se propone no es la única posible, y existen diversas alternativas en la literatura; al mismo tiempo, dependiendo del tipo de herramientas disponibles para la implementación de estas estructuras, será necesario estudiar en cada caso cuál es la más ventajosa de todas ellas. Por ejemplo, Lakhani [LAK94] recoge los datos para m=64 (con lo que las entradas se restringen a 6 bits) con un proceso MOSIS de 2 μm. En esta propuesta, los decodificadores y la ROM, dado su reducido tamaño, se han implementado en el mismo chip; de esta forma, el área del chip resultó ser de 1815×1813 μm, con un 75% de la misma dedicada a interconexiones.

Una propiedad interesante de la estructura de la figura 3.3b es que permite implementar con un coste añadido mínimo la operación de resta. Con esta estructura sólo

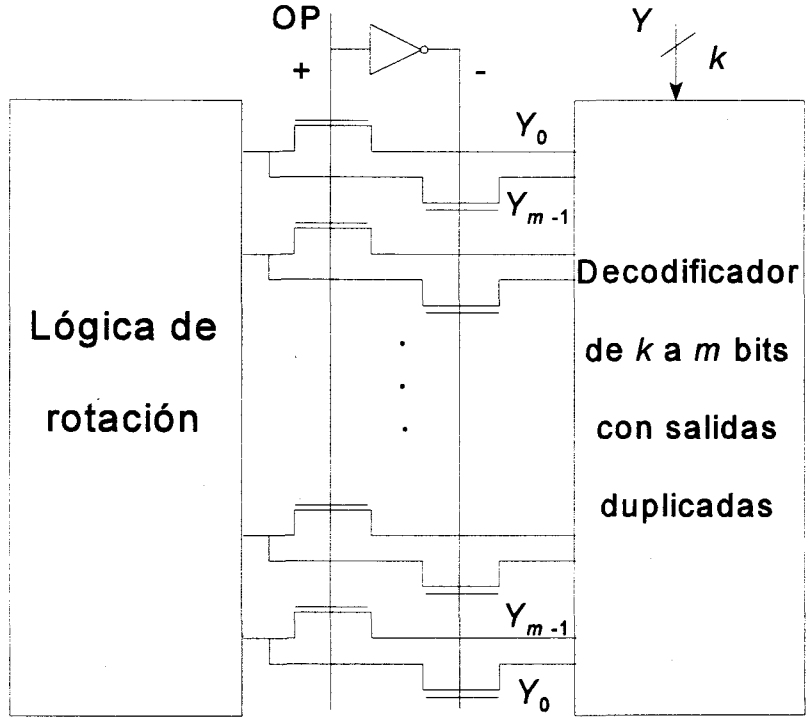


Figura 3.5. Modificación del control para implementar la resta.

es necesaria una pequeña modificación de la lógica de control del desplazador; recordemos que la ROM de la figura 3.3b almacena los elementos del grupo módulo m en orden decreciente, con lo que la suma se realiza rotando la cadena que proporciona el decodificador de X a la izquierda; de este modo, sólo hay que rotar esta cadena a la derecha para obtener en la cadena $\{Z_{m-1} \cdots Z_0\}$ la correspondiente a la resta de los operandos. Para incluir la resta en la estructura que se está comentando solamente hay que implementar la rotación a derecha o izquierda según la operación deseada. Para evitar la modificación del desplazador de la figura 3.4 parece más razonable y sencillo alterar la lógica de control, dado que la rotación de la entrada hacia la derecha en Y posiciones equivale a la rotación hacia la izquierda en m-Y posiciones; esto equivale a duplicar el conjunto de líneas de control, intercalando $\{Y_0\ Y_1 \cdots Y_{m-1}\}$ con $\{Y_{m-1}\ Y_{m-2} \cdots Y_0\}$, lo que puede hacerse duplicando las salidas del decodificador de Y e introduciendo transistores de paso, tal y como refleja la figura 3.5, de manera que debe aumentar el espacio dedicado a interconexiones (no se olvide que hay que duplicar las salidas del decodificador y

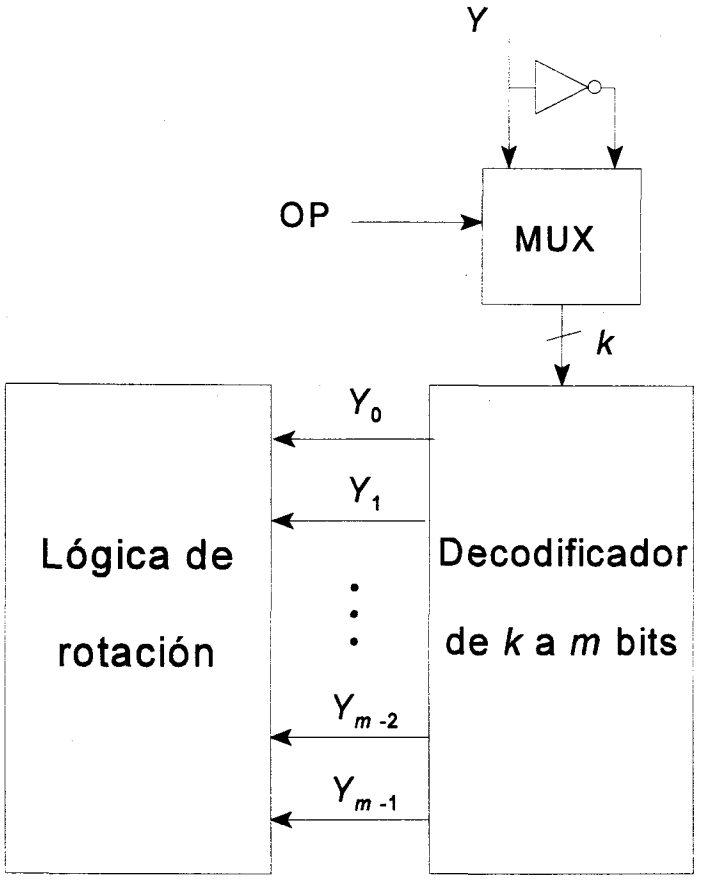


Figura 3.6. Otra propuesta para la resta.

"cruzar" o "invertir" espacialmente uno de los grupos de salidas, añadiendo los transistores de selección).

Existe una segunda posibilidad para la implementación de la resta con el método de rotación-selección, que se muestra en la figura 3.6. Esta estructura parte del hecho de que el complemento a uno de Y aplicado sobre el decodificador produce el conjunto de entradas al desplazador adecuado para realizar la resta; de esta forma, basta aplicar la señal OP de la figura 3.5 a un conjunto de multiplexores que tengan como entradas los bits de Y y sus complementos, y cuya salida sirve de entrada al decodificador.

3.1.3. Sumador RNS para módulos grandes

Conforme crece el valor de m, los circuitos y algoritmos presentados en las secciones 3.1.1 y 3.1.2 van presentando cada vez más inconvenientes; esto es especialmente reseñable para el método de rotación-selección, ya que el área total de la lógica de desplazamiento crece como m^2. A continuación se va a presentar una propuesta alternativa [LAK94] para el caso en que m posea un elevado valor y sea compuesto, es decir:

$$m = m' m''$$

con m' y m'' enteros. Dado $Z \in \mathbf{Z}_m$, existen dos enteros q_z y r_z tales que:

$$Z = q_z m' + r_z$$

donde $q_z \in [0, m''-1]$ y $r_z \in [0, m'-1]$. La pareja (q_z, r_z) recibe el nombre de componentes de Z. Cuando m' y m'' son primos entre sí, las componentes de Z son únicas, lo que permite enunciar el teorema siguiente:

Teorema 3.1: *Dados dos enteros* X *e* Y *en el rango* [0, m-1] *a través de sus componentes* (q_x, r_x) *y* (q_y, r_y), *respectivamente, su suma módulo* m *puede expresarse a través de sus componentes como:*

$$(X+Y) \bmod m = \left((q_x + q_y + \mathrm{OVF}) \bmod m'', (r_x + r_y) \bmod m' \right) \tag{3.2}$$

donde:

$$\mathrm{OVF} = \begin{cases} 0 & \text{si } r_x + r_y < m' \\ 1 & \text{en otro caso} \end{cases} \tag{3.3}$$

Demostración: por definición de componentes:

$$\begin{aligned} X+Y &= (q_x + q_y)\, m' + (r_x + r_y) = \\ &= (q_x + q_y + \mathrm{OVF})\, m' + (r_x + r_y) \bmod m' \end{aligned} \tag{3.4}$$

Para probar la ecuación (3.2), primero se considera que $(X+Y)<m$; entonces, de (3.4) se deduce que $(q_x+q_y+\mathrm{OVF})<m''$ y, por tanto $(q_x+q_y+\mathrm{OVF}) = (q_x+q_y+\mathrm{OVF}) \bmod m''$, lo que prueba (3.2) para este caso. Por otra parte, si $(X+Y)\geq m$:

$$\begin{aligned} (X+Y) \bmod m &= (X+Y) - m = \\ &= (q_x + q_y + \mathrm{OVF} - m'')\, m' + (r_x + r_y) \bmod m' \end{aligned}$$

Dado que $(X+Y)-m<m$, de (3.4) se deduce que $(q_x+q_y+\mathrm{OVF}-m'')<m''$ y, por tanto, $(q_x+q_y+\mathrm{OVF}-m'') = (q_x+q_y+\mathrm{OVF}) \bmod m''$, que de nuevo prueba (3.2), q.e.d.

Del teorema anterior se deduce que el sumador en módulo m puede construirse a partir de sumadores en módulo m' y m'', una vez que se disponga del hardware necesario para implementar la función OVF.

En primer lugar es necesario notar que el desbordamiento, es decir, $(X+Y)\geq m$, ocurre si y sólo si $0\leq(X+Y) \bmod m<X$, por lo que la representación de los operandos en cadena binaria que vimos en la sección anterior puede ser útil en la detección del desbordamiento. Aunque ésta es una operación complicada, tal y como se mostró en el apartado 2.1.2, la representación que implica el método de rotación-selección facilita esta

tarea; en concreto, el desbordamiento se produce cuando la posición del bit que es uno en la cadena que representa $(X+Y)$ mod m está a la derecha del bit que es uno en la cadena que representa a X. De este modo, sólo es necesario detectar cuándo se produce esta condición, lo que puede realizarse fácilmente, para generar la señal OVF del teorema anterior.

Por tanto, la suma de OVF en la expresión (3.2) puede hacerse añadiendo una fila más al desplazador de $m''\times m''$, ya que consiste en sumar uno (realizar un desplazamiento más sobre la cadena resultado), estando esta fila controlada directamente por OVF, como muestra la figura 3.7. La lógica para generar la señal de OVF debe detectar si el bit a uno está en el resultado a la derecha o no respecto del bit del operando a desplazar; para ello, como ya se ha citado, sólo hay que modificar ligeramente la lógica de rotación, lo que puede hacerse con el esquema de desplazador de barril $m\times m$ que se presenta en la figura 3.8, con dos conjuntos de líneas de salida, $\{b_0,..., b_{m-1}\}$ y $\{c_0,..., c_{m-1}\}$; las celdas (representadas con un círculo para simplificar la figura) que están por encima o sobre la diagonal principal del desplazador llevan su salida al primer conjunto; si (i,j) denota las coordenadas de la celda sobre la figura, estas celdas corresponden a $(i+j)<m$. Las celdas que quedan por debajo de la diagonal, $(i+j)\geq m$, proporcionan la salida sobre el segundo conjunto. Por tanto, el desbordamiento se produce cuando la operación OR de todas las líneas c_i es 1, por lo que una simple puerta OR de m entradas (no reflejada en la figura 3.8) basta para sintetizar la función OVF, además de duplicar las salidas del desplazador. Por último, la salida i-ésima Z_i del desplazador se calcula con la suma lógica (operación OR) de b_i y c_i, tal y como se muestra en el esquema de la figura 3.8. Por tanto, el desplazador de $m'\times m'$ celdas con la función OVF de la figura 3.7 correspondería al esquema anterior, mientras que el de $m''\times m''$ no necesitaría ningún tipo de modificación respecto a la figura 3.4 (salvo una fila adicional para implementar la suma de OVF).

Tras este desarrollo es fácil ver la reducción del área necesaria para implementar el sumador, que con el método de rotación-selección sería proporcional a $m^2=(m')^2(m'')^2$, mientras que con este esquema, despreciando la lógica de decodificación y de generación de OVF frente a la de rotación, el área es proporcional a $(m')^2+(m'')^2$, lo que supone una reducción importante. Sin embargo, supone una penalización en cuanto es necesario un

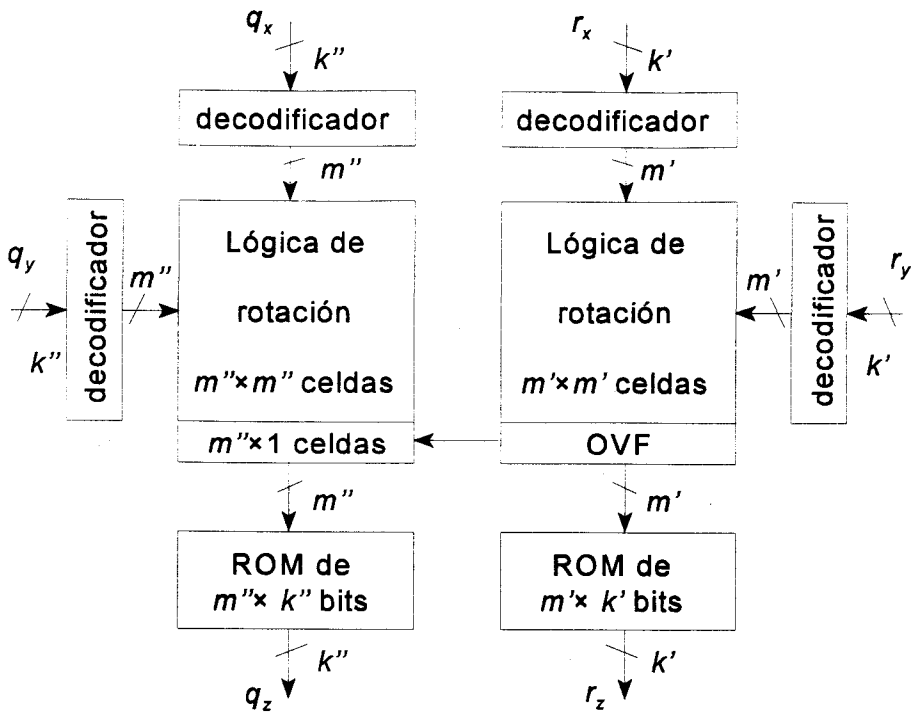


Figura 3.7. Diagrama de bloques para un sumador RNS para módulos grandes.

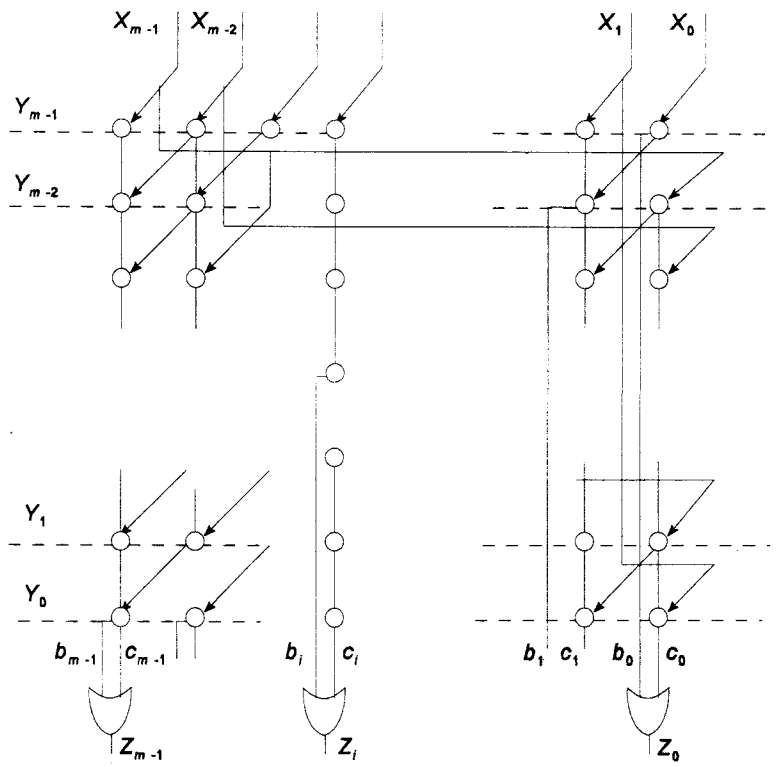


Figura 3.8. Modificación para detección de desbordamiento.

cálculo previo de las componentes de cada operando, lo que puede realizarse fácilmente con una ROM de $m\times(k'+k'')$ bits. También es necesario el cálculo del valor de la suma a partir de sus componentes, que implica el uso de una ROM de $(m'+m'')\times k$ bits donde k es el número de bits necesario para representar los m enteros con los que se trabaja. La penalización no sólo se refiere al coste añadido de las memorias, sino también al tiempo de cálculo, ya que si se decide emplear una ROM para cada entrada y otra para la salida, el cálculo sigue siendo paralelo, pero hay que añadir el retardo adicional de las memorias; por contra, si sólo se usa una ROM, habrá que dotar al sumador de cierta secuencialidad, para obtener las componentes de los dos sumandos unas tras otras.

Otro hecho interesante es que el teorema 3.1 puede generalizarse si m consta devarios factores; si $m = m_0\, m_1\, m_2$ y $Z\in[0,\ m\text{-}1]$, las componentes de Z son la tripleta $(q_z,\ r_z,\ s_z)$, siendo:

$$Z = q_z\, m_1\, m_2 + r_z\, m_2 + s_z \tag{3.5}$$

donde $q_z\in[0,\ m_0\text{-}1]$, $r_z\in[0,\ m_1\text{-}1]$ y $s_z\in[0,\ m_2\text{-}1]$. La generalización del teorema es:

Teorema 3.2: *Dados dos enteros* X *e* Y *en el rango* [0, m-1] *a través de sus componentes* $(q_x,\ r_x,\ s_x)$ *y* $(q_y,\ r_y,\ s_y)$, *respectivamente, su suma en módulo* m *puede expresarse a través de sus componentes como:*

$$(X+Y) \bmod m =$$
$$= \left((q_x + q_y + \mathrm{OVF}_2) \bmod m_0,\ (r_x + r_y + \mathrm{OVF}_1) \bmod m_1,\ (s_x + s_y) \bmod m_2\right) \tag{3.6}$$

donde OVF_1 *corresponde a* $(s_x+s_y) \bmod m_2$, *y* OVF_2 *a* $(r_x+r_y+\mathrm{OVF}_1) \bmod m_1$.

Este teorema puede fácilmente extenderse a cualquier número de factores m_i, sin más que extender la analogía entre los teoremas 3.1 y 3.2.

El uso del esquema de la figura 3.7, como se ha dicho, se ve penalizado por el cálculo de las componentes, que puede evitarse para ciertos valores de los módulos, lo que aumenta el interés de la propuesta [LAK94]. Por ejemplo, si m se elige como el producto $s2^n$, con s y n enteros, las componentes pueden obtenerse directamente de la representación

binaria de los operandos; en concreto, si s necesita t bits para su representación binaria, los n bits menos significativos de la representación binaria de $(t+n)$ bits del operando X constituyen r_x, mientras que los t bits más significativos forman q_x. En este caso, es obvio que $m'=s$ y $m''=2^n$.

De este modo, se observa que el método de rotación-selección es útil para la implementación de la suma y la resta módulo m para valores no muy elevados de m; además, es importante reseñar que no depende en modo alguno de la forma concreta de m, y el método es válido para cualquier tipo de módulo. Se puede demostrar [LAK94] que este método supone un compromiso (con respecto a la relación área-velocidad) entre la implementación directa con tablas de consulta y los sumadores basados en sumadores binarios. Además, es posible extender este método para módulos grandes cuando m es compuesto.

3.2. MULTIPLICADORES EN ARITMÉTICA DE RESIDUOS

Tras revisar algunas propuestas para la construcción de sumadores en el RNS, en esta sección se van a mostrar algunas propuestas para la implementación de la multiplicación módulo m, indicativas de las arquitecturas empleadas en los últimos años para la realización de esta operación.

3.2.1. Multiplicador para el conjunto de módulos $\{2^n-1, 2^n, 2^n+1\}$

La primera propuesta que se presenta [TAY82] hace uso de multiplicadores VLSI de $n \times n$ bits con el conjunto de módulos $\{2^n-1, 2^n, 2^n+1\}$. Para el módulo potencia de dos, sean $m=2^n$ y $k=n/2$; si $x \in Z_m$, entonces:

$$x = 2^k x_{HI} + x_{LO} \qquad \text{con} \quad x_{HI},\ x_{LO} \in Z_{2^k} \tag{3.7}$$

Por tanto, x_{HI} y x_{LO} son enteros positivos de k bits. Si $y \in Z_m$, con una expresión similar a la (3.7):

$$z = (xy) \bmod m = \left(2^n a + b + 2^k (c + d)\right) \bmod m \tag{3.8}$$

donde:

$$\begin{aligned} a &= x_{HI} y_{HI};\ 0 \le a \le (2^k - 1)^2 < 2^n - 1 \\ b &= x_{LO} y_{LO};\ 0 \le b \le (2^k - 1)^2 < 2^n - 1 \\ c &= x_{HI} y_{LO};\ 0 \le c \le (2^k - 1)^2 < 2^n - 1 \\ d &= x_{LO} y_{HI};\ 0 \le d \le (2^k - 1)^2 < 2^n - 1 \\ v &= c + d;\ 0 \le v \le 2(2^k - 1)^2 \end{aligned} \tag{3.9}$$

Bajo la hipótesis de que $m=2^n$, es claro que $2^k=2^n/2^k$, y entonces:

$$z = \left(\left(2^n a\right) \bmod 2^n + b \bmod 2^n + \left(2^n (c + d)/ 2^k\right) \bmod 2^n \right) \bmod 2^n \tag{3.10}$$

A partir de la representación binaria de $v=(c+d)$ (expresión (3.9)), de $n+1$ bits, $v/2^k$ puede formarse sin más que definir el dígito binario que precede al bit de valor 2^k-1; es decir, si $v/2^m = V_{int}+V_{frac}$, donde V_{int} es la parte entera de $v/2^m$ y V_{frac} es la parte fraccionaria, entonces:

$$\left(2^n v/2^k\right) \bmod 2^n = \left(2^n (V_{int} + V_{frac})\right) \bmod 2^n = \left(2^n V_{int}\right) \bmod 2^n + \left(2^n V_{frac}\right) \bmod 2^n$$

El cálculo de $(2^n V_{int}) \bmod 2^n$ puede resultar una operación compleja con los métodos digitales convencionales; sin embargo, no es éste el caso, ya que V_{int} es una palabra de k bits, con $k=n/2$. Por ejemplo, si se desea un módulo de 24 bits, entonces $k=12$ y una memoria de 4K×1 bits puede almacenar $(2^n V_{int}) \bmod 2^n$; posteriormente, los productos parciales pueden combinarse empleando un sumador módulo 2^n para formar z.

En lo que respecta a $m=2^n$-1, si:

$$\begin{aligned} 2^n &= (2^n - 1) + 1 \\ 2^k &= 2^n / 2^k = (2^n - 1)/2^k + 1/2^k \end{aligned} \tag{3.11}$$

entonces:

$$z = \left(\left((2^n - 1) a + a \right) + b + \left(\left(2^n - 1\right)\left(v/2^k\right) + v/2^k \right) \right) \bmod m \tag{3.12}$$

de donde se deduce que $a \bmod m = a$ y $b \bmod m = b$, y $v/2^k = V_{int} + V_{frac}$, con:

$$\left(\left(2^n - 1\right) v/2^k\right) \bmod m = \left(\left(2^n - 1\right) \times \left(V_{int} + V_{frac}\right)\right) \bmod m = \left(\left(2^n - 1\right) V_{frac}\right) \bmod m \quad \textbf{(3.13)}$$

Usando operaciones con tablas y una unidad de memoria de 2^m palabras se puede implementar directamente el mapeo modular. Los términos de producto parcial se combinan en un sumador módulo m.

Por último, para $m = 2^n+1$ se requieren algunas modificaciones especiales; en particular, no todos los residuos pueden codificarse en una palabra de n bits; en concreto, $x = 2^n$ requiere $n+1$ bits; sin embargo, este valor es de la forma $100...00_b$, que permite una fácil detección, por lo que se puede añadir la lógica necesaria para la detección de 2^n en alguna de las entradas y la síntesis de la salida adecuada en dicho caso. Concretamente, si $x = 2^n$ e $y < 2^n$, entonces $(xy) \bmod m = (-y) \bmod m$, y si $x = y = 2^n$, $(xy) \bmod m = 1$. Por tanto, una arquitectura capaz de realizar la multiplicación para un módulo grande, adecuada para una implementación VLSI, sería la de la figura 3.9, en la que las diferentes operaciones que se realizan se recogen en la tabla 3.2; en ésta, la última columna indica el hardware que genera dichos valores (para cada módulo).

TABLA 3.2

Operación	$m=2^n$	$m=2^n+1$	$m=2^n+1$	
$S1=(2^n x_{HI} y_{HI}) \bmod m$	0	a	$-a$	multiplicador VLSI
$S2=(x_{LO} y_{LO}) \bmod m$	b	b	b	multiplicador VLSI
$T1=x_{HI} y_{LO}$	c	c	c	multiplicador VLSI
$T2=x_{LO} y_{HI}$	d	d	d	multiplicador VLSI
$U=(a+b) \bmod m$	b	$(a+b) \bmod m$	$b-a$	sumador módulo m
$V=c+d$	V_{int}	V_{int}	V_{int}	sumador+registro despl.
V'	0	$+V_{int}/2^k$	$-V_{int}/2^k$	sumador+registro despl.
V_{frac}	$.V_{frac}$	$.V_{frac}$	$.V_{frac}$	sumador+registro despl.
$W1=(U+V) \bmod m$	W1	W1	W1	sumador módulo m
$W2=(p.V_{frac}) \bmod m$	W2	W2	W2	tabla de consulta
$Z=(W1+W2) \bmod m$	2	2	2	sumador módulo m

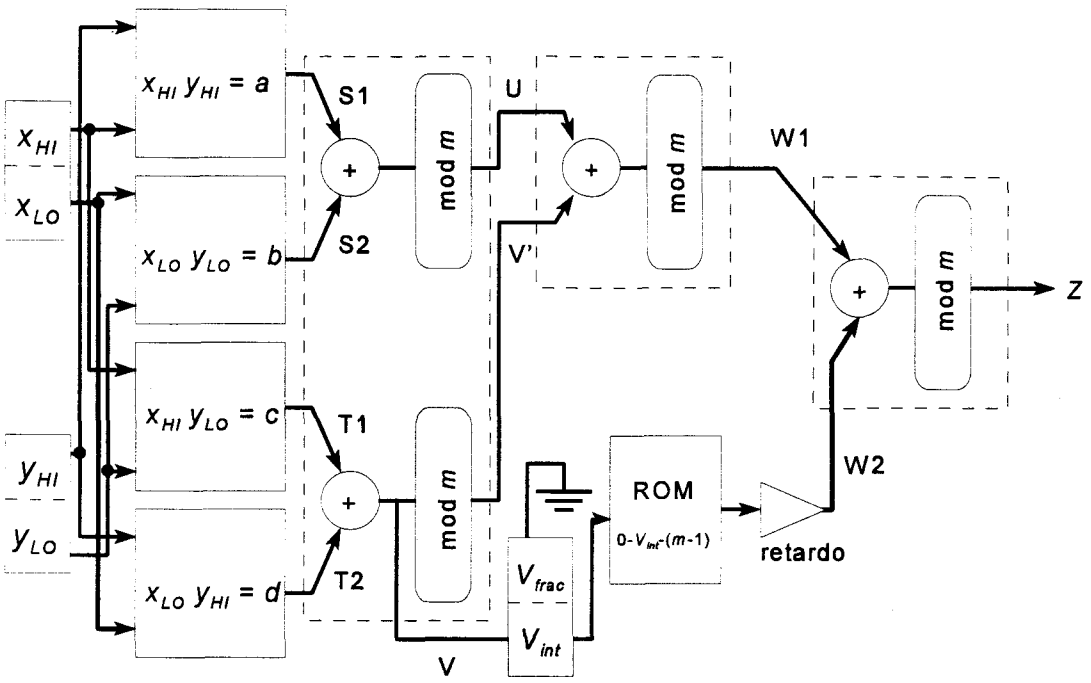


Figura 3.9. Multiplicador VLSI para el sistema numérico de residuos.

Taylor recoge en su trabajo [TAY82] una implementación física de este multiplicador, con cuatro tipos de multiplicadores VLSI comerciales, sumadores expresamente realizados para la aplicación y unidades de memoria para la implementación de las tablas. Para n=24 ó 16 bits, y con estas premisas, se puede construir un sistema multiplicador con los tres módulos citados, es decir, con un rango dinámico entre 72 y 48 bits. Esto se traduce en una velocidad de 2.5 millones de multiplicaciones por segundo en una realización secuencial, que se transforman en 10 millones de operaciones por segundo si a la estructura de la figura 3.9 se la dota de segmentación de cauce; estos datos resultan impresionantes teniendo en cuenta la fecha de la propuesta y el rango dinámico (hasta 72 bits) que puede manejarse.

3.2.2. Multiplicador para módulos grandes

También orientado a la implementación VLSI, Alia y Martinelli [ALI91] han

propuesto un nuevo algoritmo para la multiplicación en módulo m, basado en una multiplicación binaria fraccional aproximada. En concreto, sean $\alpha,\beta\in[0, m\text{-}1]$, que se representan con $b=\lceil \log m \rceil$ bits; por tanto, se requieren $2b$ bits para la representación del producto $p = \alpha \cdot \beta$; por otra parte, llamaremos $\pi = (\alpha \cdot \beta) \bmod m$, el valor que se desea calcular. Es evidente que:

$$\pi = p - k \cdot m$$

donde $k=\lfloor p/m \rfloor$, por la propia definición del residuo módulo m de p. Por tanto, el problema de realizar una multiplicación en módulo m puede reducirse a evaluar la constante k. Dado que el cálculo de p/m puede requerir un número indefinido de bits para su representación, el valor $1/m$ se aproxima por un número racional t, que se elige de manera adecuada para que el error introducido sobre la parte entera de k sea fácilmente corregido independientemente del valor de p; en concreto, k se sustituye por un entero K que se obtiene de una manera más sencilla y que sólo puede tomar los valores k y k-1. Si t es el número que se obtiene al truncar $1/m$ en los primeros r bits correspondientes a la parte fraccional, entonces:

$$t \le 1/m < t + 2^{-r} \qquad \textbf{(3.14)}$$

y si multiplicamos por p:

$$p \cdot t \le p/m < p \cdot t + p \cdot 2^{-r}$$

de forma que el máximo error generado al aproximar p/m por $p \cdot t$ está acotado por:

$$E = p \cdot 2^{-r}$$

Si se impone que E sea menor que la unidad, ha de satisfacerse que $r \ge 2b$. En efecto, bajo este supuesto y considerando $p<2^{2b}$, se tiene que:

$$E = p \cdot 2^{-r} < 2^{2b} \cdot 2^{-r} \le 1$$

Si se denomina K a $\lfloor p \cdot t \rfloor$, se puede escribir:

$$k = \lfloor p/m \rfloor = \lfloor p \cdot t + E \rfloor = \lfloor K + \mathrm{Frac}(p \cdot t) + E \rfloor =$$
$$= K + \lfloor \mathrm{Frac}(p \cdot t) + E \rfloor = K + E'$$

donde Frac(z) indica la parte fraccional del número real z. Dado que $0 \leq \mathrm{Frac}(p \cdot t) + E < 2$, es claro que $E' \in \{0, 1\}$. Finalmente, el producto módulo m puede obtenerse como:

$$\pi = p - K \cdot m - E' \cdot m \qquad \textbf{(3.15)}$$

En lo que se refiere a la parte computacional del algoritmo, el cálculo de π implica el cálculo de $p\text{-}K \cdot m$, así como el test de su valor; si éste es menor que m, es el correcto, y si no, debe restársele m para obtener el valor correcto. Merece la pena observar que el valor resultante de la resta $p\text{-}K \cdot m$ es siempre positivo y menor que $2m$, con lo que b+1 bits son suficientes para su representación.

Como ejemplo, se va a aplicar todo esto a un ejemplo concreto. Sea el sistema de residuos definido por los módulos m_1=5, m_2=7, m_3=11 y m_4=13, en el que se va a realizar el producto de los números:

$$37 \equiv \{010, 010, 0100, 1011\}$$
$$12 \equiv \{010, 101, 0001, 1100\}$$

Como se ve, el residuo correspondiente a cada módulo se expresa con b_i bits, siendo b_1=3, b_2=3, b_3=4 y b_4=4. Al multiplicar se obtiene que:

$$p_1 \equiv \alpha_1 \beta_1 = 010 \times 010 = 000100$$
$$p_2 \equiv \alpha_2 \beta_2 = 010 \times 101 = 001010$$
$$p_3 \equiv \alpha_3 \beta_3 = 0100 \times 0001 = 00000100$$
$$p_4 \equiv \alpha_4 \beta_4 = 1011 \times 1100 = 10000100$$

Si se escoge r_1=6, r_2=6, r_3=8 y r_4=8, los valores de las fracciones $1/m_i$ truncados sobre r_i bits pueden elegirse, de acuerdo con (3.14), como las constantes:

$$t_1 = .001100, t_2 = .001001, t_3 = .00010111, t_4 = .00010011$$

con lo que los valores de K_i son:

$$K_1 \equiv \lfloor p_1 \cdot t_1 \rfloor = 000100 \times .001100 = 0000$$
$$K_2 \equiv \lfloor p_2 \cdot t_2 \rfloor = 001010 \times .001001 = 0001$$
$$K_3 \equiv \lfloor p_3 \cdot t_3 \rfloor = 00000100 \times .00010111 = 00000$$
$$K_4 \equiv \lfloor p_4 \cdot t_4 \rfloor = 10000100 \times .00010011 = 01001$$

Por tanto:

$$p_1 - K_1 \cdot m_1 = 000100 - (0000 \times 101) = 0100$$
$$p_2 - K_2 \cdot m_2 = 001010 - (0001 \times 111) = 0011$$
$$p_3 - K_3 \cdot m_3 = 00000100 - (00000 \times 1011) = 00100$$

y dado que estos valores son menores que m_1=5, m_2=7 y m_3=11, es claro que π_1=100, π_2=011 y π_3=0100; para π_4:

$$p_4 - K_4 \cdot m_4 = 10000100 - (01001 \times 1101) = 10000100 - 001110101 = 01111$$

que es mayor que m_4=13, por lo que se necesita una resta adicional (E'_4=1), con lo que π_4=01111 - 01101 = 0010. El conjunto {4, 3, 4, 2} evaluado en el sistema de residuos en el que se está trabajando representa el producto $X \times Y$=37×12=444.

Este algoritmo puede realizarse con la estructura de la figura 3.10, en la que los enteros α y β, de b bits, se multiplican en MUL1, obteniendo el valor p de $2b$ bits; este valor es multiplicado por t, la aproximación de $1/m$, en MUL2, que proporciona como resultado R, un número de $4b$ bits. R contiene en los bits $\{3b,..., 2b+1, 2b\}$ la representación de K, que se multiplica en MUL3 por $-m$ a fin de obtener $C=-K \cdot m$. El sumador ADD1 genera $D=p-K \cdot m$, con $b+1$ bits para su representación; el sumador ADD2 genera $D-m$ con su signo correspondiente, de forma que el resultado de la multiplicación será D si $D-m<0$, y $D-m$ si $D>m$. Es preciso señalar que, dado que D se representa con $b+1$ bits, los bits $\{b+2,..., 2b-1\}$ de C y p no se tienen en cuenta en el sumador ADD1.

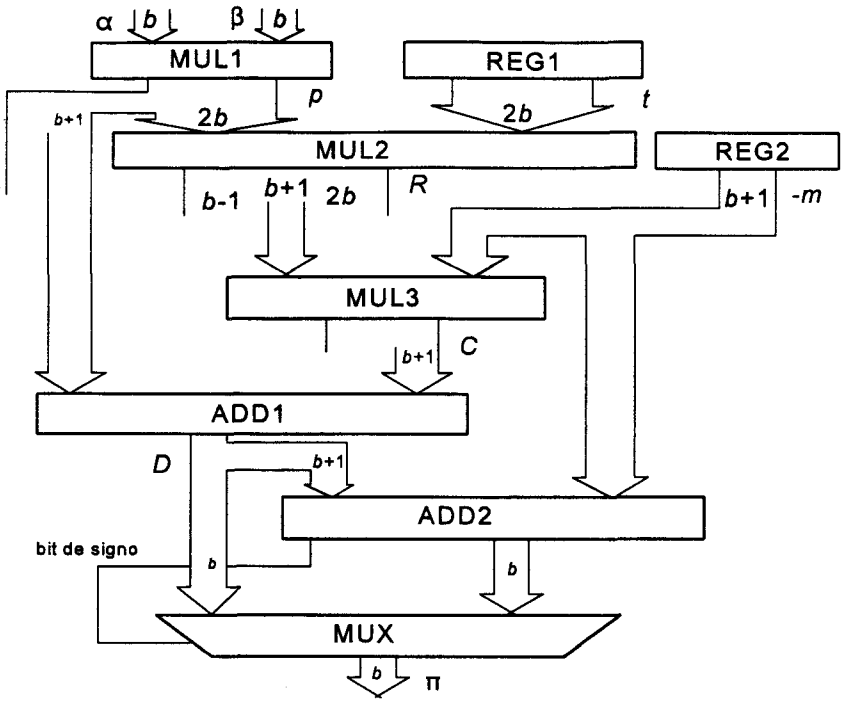


Figura 3.10. Diagrama de bloques para un multiplicador módulo m.

Esta estructura es aplicable al RNS, y puede proporcionar unas buenas prestaciones para un rango dinámico considerable. Alia y Martinelli [ALI91] proponen la elección de diez módulos de 15 bits, m_1=32047, m_2=31879, m_3=31679, m_4=31069, m_5=31067, m_6=30883, m_7=30625, m_8=30551, m_9=30427 y m_{10}=29478, siendo:

$$M = \prod_{i=1}^{10} m_i \approx 8.1 \times 10^{44}$$

que permite el producto de enteros hasta 2.8×10^{22}. Este multiplicador requiere diez multiplicadores como el de la figura 3.10, que trabajarán en paralelo, con lo que la velocidad de operación vendrá determinada por la de la unidad más lenta (en buena lógica, y dada la similitud de los submódulos, la velocidad de los diez multiplicadores será similar). Con este valor de b=15, MUL1, MUL2 y MUL3 se implementan con multiplicadores binarios 16×16, 32×32 y 16×16, respectivamente. El multiplicador 32×32 puede implementarse fácilmente con dos multiplicadores 16×16 y un sumador de 48 bits, como muestra la figura 3.11. Dado que los bits en las posiciones 16 a 29 de t son cero, ya que $1/m < 2^{-14}$, el cálculo de K se reduce en la práctica a una multiplicación 16×16 y una suma. Empleando multiplicadores comerciales de unos 50 ns, como el multiplicador de

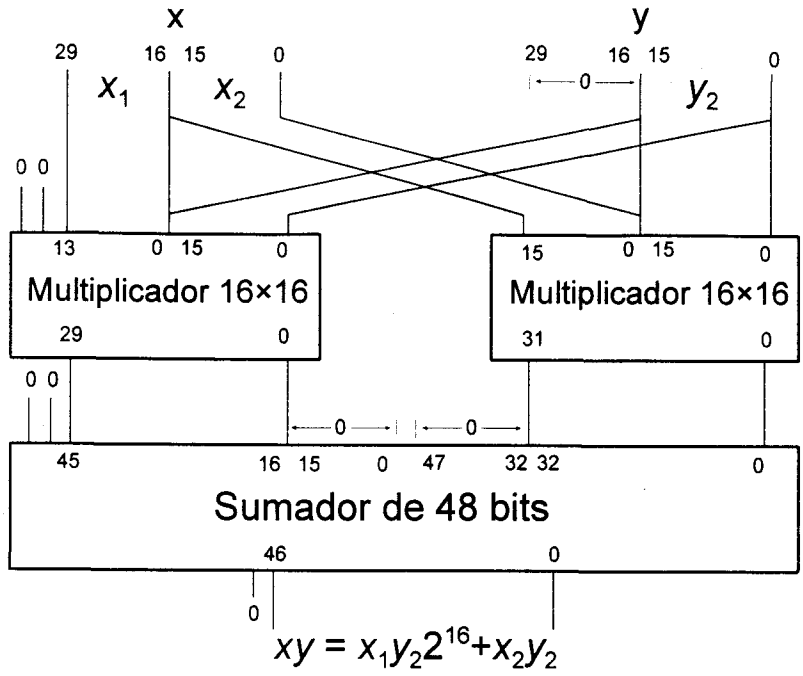


Figura 3.11. Multiplicador binario 32×32.

enteros 16×16 WTL2516C de Weitek (con un tiempo de respuesta de 38 ns), el tiempo total de multiplicación para cada uno de los submódulos se sitúa entre 150 y 200 ns. Lo que es más, si se dota a la estructura de segmentación de cauce, puede conseguirse un tiempo entre operaciones consecutivas de 70 ns. Estos resultados, que proporcionan hasta 14 millones de operaciones por segundo (con cauce segmentado, y 5 millones de operaciones por segundo si la ejecución es secuencial), no dependen del número de módulos, y se pueden añadir cuantos deseemos empleando la misma estructura para aumentar el rango dinámico, siempre que no excedan estos módulos de 32767.

3.2.3. Multiplicadores de cuerpos de Galois

Los multiplicadores expuestos en los apartado 3.2.1 y 3.2.2 están orientados a implementaciones VLSI, empleando algoritmos relativamente complejos que hacen uso de unidades aritméticas convencionales. Llegados a este punto, vamos a pasar a otro tipo de implementaciones con una filosofía totalmente distinta [JUL80, RAD91, GAR97a], de forma que el hardware empleado es sustancialmente más sencillo; naturalmente, el

inconveniente de estas estructuras es que están limitadas a módulos primos, ya que se basan en las propiedades algebraicas de los cuerpos finitos (v. apéndice A). Este tipo de implementación presenta un gran interés de cara a aplicaciones muy específicas, como son los algoritmos para el procesamiento de señal basados en el uso de transformadas teóricas numéricas (NTT: Number Theoretic Transforms) [BOU95], o más generalmente, transformadas discretas de Fourier generalizadas, GDFT (Generalized Discrete Fourier Transforms); estas transformadas son formalmente similares a la FFT (v. sección 2.2), salvo que se calculan sobre cuerpos de Galois finitos [SZA67] y son de la forma:

$$F_k = \sum_{i=0}^{N-1} f_i \, \alpha^{ik} \tag{3.16}$$

donde α es la raíz N-ésima de la unidad en GF(M), con la restricción de que existe el inverso multiplicativo de N. Aunque el dominio de la transformada no tiene uso conocido, posee la misma propiedad de convolución cíclica que la DFT clásica, por lo que puede usarse en operaciones de filtrado y correlación. Además, al realizarse el cálculo sobre un cuerpo de Galois, no hay errores en el resultado final, con sólo satisfacer que la operación de convolución tenga una cota superior:

$$\sum_{n=0}^{N-1} x_n \, y_{k-n} < M$$

suponiendo que las secuencias $\{x_n\}$ e $\{y_n\}$ están formadas por enteros positivos; esta condición asegura que el resultado es correcto y no el residuo en módulo M del mismo. Tradicionalmente se han usado los números primos de Fermat[1] y de Mersenne[2] para M, a fin de facilitar el cálculo usando aritmética binaria; por el mismo motivo, α se ha restringido a potencias de dos. Dado que existe una relación directa entre M y N (se requiere un valor grande de M para valores grandes de N), las NTTs se han implementado sobre GF(M^2). También se ha sugerido [KIN74] implementarlas en anillos isomórficos a una suma directa de cuerpos de Galois:

---

[1] números primos de la forma 2^{2n}-1, con n entero.

[2] números primos de la forma 2^n-1, con n entero.

$$R \cong GF\left(m_1{}^n\right) \oplus GF\left(m_2{}^n\right) \oplus \cdots \oplus GF\left(m_L{}^n\right)$$

donde los m_i son primos y n representa el grado de la extensión de los cuerpos. Es, pues, lógico pensar en una implementación basada en el RNS; el uso de este sistema con arquitecturas basadas en ROM permite que la selección de α y m_i se base exclusivamente en consideraciones sobre la teoría de números, sin ningún tipo de restricción debida a la necesidad de una implementación binaria de alta velocidad. Sin embargo, todavía existe la necesidad de seleccionar valores relativamente grandes de m_i por los requerimientos para una longitud razonable de la transformada ($N \geq 32$).

El principal inconveniente de lo dicho hasta este momento se centra en la cantidad de memoria requerida. Para un módulo $m_i \leq 32$, las operaciones de multiplicación y suma en dicho módulo pueden llevarse a cabo consultando el resultado en una ROM de 1K×8 bits; con la misma filosofía, si $32 < m_i \leq 64$, se necesitaría una ROM de 4K×8; para $64 < m_i \leq 128$, la memoria habría de ser de 16K×8 bits. Dado que en las aplicaciones usuales, como las NTTs, los módulos m_i a usar se sitúan por regla general en éste último rango, el uso de una memoria de 16K×8 bits no parece muy eficiente para realizar la multiplicación. Pero se puede incrementar la eficiencia de la multiplicación haciendo uso del isomorfismo (v. sección A.3, propiedad A.1) entre el grupo multiplicativo $\{1, 2, 3, \cdots, p\text{-}1\}$ con multiplicación módulo p, y el grupo aditivo $\{0, 1, 2, \cdots, p\text{-}2\}$ con suma módulo p-1, siendo p primo; de esta forma, si:

$$\forall\ q_n \in \{1, \cdots, p-1\} \quad \exists \quad i_n \in \{0, 1, \cdots, p-2\} \quad / \quad q_n = g^{i_n}$$

con p primo, se puede hacer:

$$\left| q_j\, q_k \right|_p = g^{\left| i_j + i_k \right|_{p-1}} \tag{3.17}$$

donde g es el llamado elemento primitivo. De este modo, la multiplicación módulo p se reduce a una suma módulo p-1; esta aproximación se conoce como ***cálculo de índices***, ya que el valor i_n recibe generalmente el nombre de ***índice*** de q_n. El esquema básico para la multiplicación se refleja en la figura 3.12, donde los índices de los operandos se hallan

almacenados en sendas memorias (ROM1 y ROM2), al tiempo que otra ROM (señalada como ROM3) proporciona el resultado correspondiente a la suma de los índices, que es la multiplicación de las entradas, según (3.17).

3.2.4. Aproximación 2*p* para multiplicación por cálculo de índices

La suma de índices puede acelerarse usando submódulos primos entre sí y cuyo producto sea mayor que 2*p* [JUL80], lo que se conoce como ***aproximación* 2p**; esto supone una mejora sustancial, ya que podemos realizar la suma en módulos sustancialmente diferentes de *p*, por lo que se pueden buscar los valores más adecuados para facilitar la operación, incluso a costa de añadir el hardware necesario para generar los residuos en estos submódulos, así como para obtener el valor de la suma en módulo *p*-1 a partir de la representación en residuos. En realidad, estos dos últimos inconvenientes no son tales, ya que en la ROM1 y la ROM2 de la figura 3.12 bastaría con incluir para cada entrada la representación en los submódulos elegidos, llevando los bits correspondientes a cada uno de ellos al sumador adecuado. De la misma manera, sólo habría que aplicar los diferentes residuos de la suma sobre la ROM3 para obtener directamente el valor del

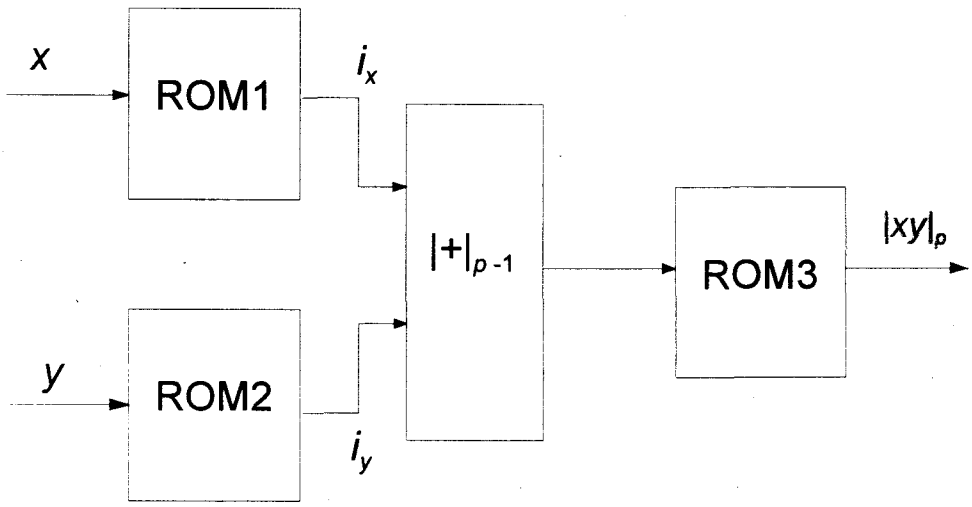


Figura 3.12. Multiplicador por cálculo de índices.

producto, con lo que también trasladamos el resultado de la suma de módulo $m_1 \times m_2$, si $m_1 \times m_2 > 2p$, a módulo p-1. Por ejemplo, si p=19, podemos hacer m_1=6 y m_2=7, al tiempo que g=2 [ABR65], de forma que las memorias ROM1 y ROM2 de la figura 3.12 contendrían lo indicado en la tabla 3.3.

TABLA 3.3

q_n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
i_n	0	1	13	2	16	14	6	3	8	17	12	15	5	7	11	4	10	9
$\lvert i_n \rvert_6$	0	1	1	2	4	2	0	3	2	5	0	3	5	1	5	4	4	3
$\lvert i_n \rvert_7$	0	1	6	2	2	0	6	3	1	3	5	1	5	0	4	4	3	2

Por tanto, para cada entrada se obtendrían los dos residuos del i_n correspondiente; estos residuos se aplican de manera adecuada sobre los sumadores en cada submódulo y los resultados se concatenan para actuar como dirección de la ROM3. Puesto que hay combinaciones que no aparecerán nunca (en el ejemplo anterior, para m_1=6, las combinaciones 101_b y 111_b no son válidas y no deben aparecer en condiciones normales), las direcciones pueden aprovecharse para la detección de errores o para simplificar el hardware.

Como ejemplo, se va aplicar esta aproximación al multiplicador módulo 19. Dado que el índice de cero no existe y la multiplicación por cero en este esquema no es posible (será preciso añadir el hardware necesario, que es muy sencillo, para la detección del cero en uno de los multiplicandos), se puede emplear el valor 111_b, que sería incorrecto para los dos submódulos propuestos, 6 y 7, como índice de cero. De esta forma, si los sumadores se implementan con memorias, para llevar a cabo la multiplicación por cero podemos dar como salida 111_b en el caso de que alguna de las entradas sea cero. Con lo dicho, el contenido de las diferentes memorias se presenta en la figura 3.13, en la que se ilustra también la multiplicación de x=12 e y=17, siendo $\lvert 12 \times 17 \rvert_{19} = 14$. Asumiendo que $i(17) = (4, 3)$ en los submódulos elegidos, 6 y 7, respectivamente, la figura 3.13 muestra la evolución de los datos a lo largo del proceso de multiplicación. Evidentemente, se puede modificar este esquema, de forma que el valor de i_n para cada una de las entradas, o mejor dicho, sus residuos, puedan obtenerse en una sola memoria, llevando las líneas

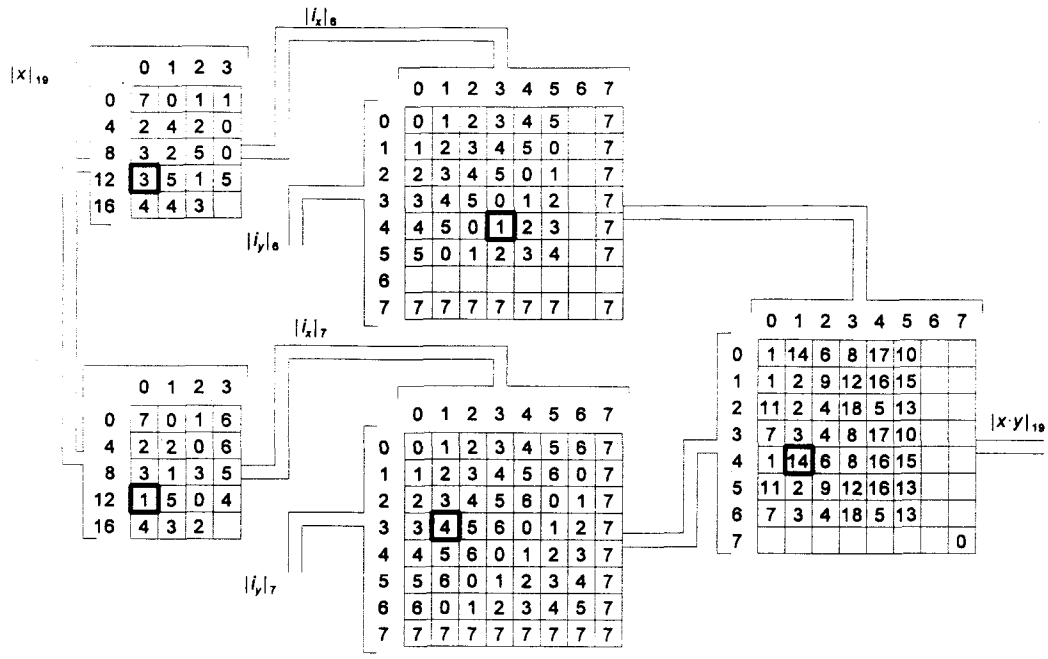


Figura 3.13. Multiplicador módulo 19 por cálculo de índices.

de salida a los sumadores adecuados. Las posiciones remarcadas son las que se van seleccionando en cada una de las memorias, de forma que si se estudia detenidamente la figura 3.13, el valor 0 en una de las entradas provoca que el resultado de las dos sumas sea $7=111_b$, con lo que la salida es necesariamente cero.

Esta aproximación presenta indudables ventajas frente al esquema de la figura 3.12, ya que el sumador se divide en dos sumadores de módulo menor, que generalmente ocuparán menos espacio o serán más rápidos; sin embargo, el inconveniente principal [GAR95] reside en el hecho de que la suma de los índices ha de realizarse módulo $2p$ (aunque se divida en submódulos menores) como mínimo, cuando lo que se desea realizar es una suma en módulo p-1; de este modo, no se aprovecha al máximo la potencialidad del cálculo de índices, consumiendo más memoria de la estrictamente necesaria. Sería, pues, deseable hallar algún mecanismo que permitiera evitar el ampliar el rango de la suma. La solución va a venir de nuevo de las propiedades de los cuerpos de Galois, de forma que la suma de los índices se realice sobre unos submódulos que formen un isomorfismo entre el grupo multiplicativo de GF(p) y la suma directa de grupos aditivos más pequeños, tal y como se comentó antes para la NTT; esto permite que el sumador de la figura 3.12 se divida en cierto número de sumadores de menor submódulo, pero sin

necesidad de imponer que el producto de los submódulos sea mayor que $2p$, como se mostrará en el siguiente apartado.

3.2.5. Aproximación isomórfica submodular

La aproximación $2p$ puede mejorarse partiendo del teorema que se enuncia a continuación:

Teorema 3.3: *Sea* $\{q_n\}=\{1, 2,..., p-1\}$ *un grupo multiplicativo e* $\{i_n\}=\{0, 1,..., p-2\}$ *su grupo aditivo asociado e isomórfico, con el isomorfismo definido por:*

$$q_n = \left| g^{i_n} \right|_p$$

donde g *es un elemento primitivo de* GF(p). *Si:*

$$p-1 = \prod_{j=1}^{r} m_j \qquad \textbf{(3.18)}$$

siendo $m_1, m_2,..., m_r$ *primos entre sí, entonces existe un isomorfismo* $f:\{i_n\}\rightarrow\{U_k\}$ *donde* $U_k\in\{U_0, U_1, ..., U_{p-2}\}$ *para todo* $k\in\{i_n\}$, *siendo:*

$$U_k = \left(|k|_{m_1}, |k|_{m_2}, \ldots, |k|_{m_r} \right)$$

Demostración: se define:

$$f(k) = U_k = \left(|k|_{m_1}, |k|_{m_2}, \ldots, |k|_{m_r} \right) \qquad \forall\, k \in \{i_n\}$$

y sean dos elementos $k_1, k_2\in\{i_n\}$. Entonces:

$$f(k_1 + k_2) = U_{k_1+k_2} = \left(|k_1 + k_2|_{m_1}, |k_1 + k_2|_{m_2}, \ldots, |k_1 + k_2|_{m_r} \right) =$$

$$= \left(|k_1|_{m_1}, |k_1|_{m_2}, \ldots, |k_1|_{m_r} \right) + \left(|k_2|_{m_1}, |k_2|_{m_2}, \ldots, |k_2|_{m_r} \right) = U_{k_1} + U_{k_2} = f(k_1) + f(k_2)$$

por la propia definición del Sistema Numérico de Residuos (v. sección 2.1), ya que $f(k)$ es la representación de k en el RNS definido por $\{m_1, m_2,..., m_r\}$, con lo que es inmediato

que f es un homomorfismo.

Por otra parte, si:

$$p-1=\prod_{j=1}^{r} m_j$$

y mcd(m_i, m_j)=1 para $i\neq j$, es decir, los submódulos son primos entre sí, el teorema del resto chino (teorema 2.1) asegura que la representación en residuos de k es única para todo $k\in\{i_n\}$, por lo que este teorema asegura que f es un isomorfismo, q.e.d.

El teorema anterior puede aplicarse al siguiente ejemplo: sea p=7, con $\{q_n\}$={1, 2, 3, 4, 5, 6} e $\{i_n\}$={0, 1, 2, 3, 4, 5}, g=3 ó g=5. Es obvio que 2 y 3 son los únicos submódulos que satisfacen que p-1=6=2×3; el isomorfismo definido por el teorema 3.3 se muestra en la tabla 3.4, con g=3.

TABLA 3.4

$\mathbf{q_n}$	1	2	3	4	5	6
$\mathbf{i_n}$	0	2	1	4	5	3
$\mathbf{U_k}$	(0, 0)	(0, 2)	(1, 1)	(0, 1)	(1, 2)	(1, 0)

De este modo, la multiplicación de x=2 e y=3 se lleva a cabo obteniendo los índices, como ya hemos visto, i_x=2 e i_y=1. Por tanto, U_x=(0, 2) y U_y=(1, 1), con lo que U_x+U_y=(1, 0), que según la tabla, corresponde a q_{xy}=6, que es el valor del producto xy.

El teorema 3.3 demuestra que la suma de índices sobre el grupo aditivo con suma módulo p-1 de GF(p) puede llevarse a cabo eficientemente como un conjunto de operaciones de suma concurrentes sobre los submódulos m_1, m_2,..., m_r, primos entre sí y satisfaciendo:

$$p-1=\prod_{j=1}^{r} m_j$$

con lo que el esquema seguiría siendo el mismo que se viene discutiendo (figura 3.12), pero transformando el sumador módulo p-1 en *r* sumadores más pequeños, y aplicando los *r* resultados a la memoria ROM3. Esta aproximación también recibe en la literatura el nombre de ***técnica de mapeo isomórfico submodular*** o ***aproximación isomórfica submodular***.

La selección de los submódulos debe realizarse según ciertos criterios que minimicen el área de silicio empleada, o en su caso, la complejidad de los componentes, al tiempo que mejoren en lo posible la velocidad de los sumadores; estos criterios para la elección de los submódulos se centran en:

a) $p-1=\prod_{j=1}^{r} m_j$;

b) $\text{mcd}(m_j,\ m_k)=1 \quad \text{para } i\neq k$;

c) minimizar $\sum_{j=1}^{r} \lceil \log_2(m_j-1) \rceil$;

con lo que el último criterio no implica más que minimizar en lo posible el tamaño de la ROM3 de la figura 3.12, la memoria en la que se realiza el cálculo inverso de los índices, así como la longitud de palabra de las memorias ROM1 y ROM2. Al mismo tiempo, minimizar $\lceil \log_2 (m_j-1) \rceil$ de forma individual lleva a poder realizar de manera más rápida y con un hardware más sencillo la suma de los residuos de los índices de ambos operandos.

Este método es aplicable a la mayor parte de los números primos *p*, ya que cualquier entero (*p*-1) satisfaciendo $(p-1)\neq 2^n$, con *n* entero, puede factorizarse en la forma que muestra la ecuación (3.18); para los enteros hasta 2887, $p=17$ y $p=257$ son las únicas excepciones. Además, esta elección de submódulos primos entre sí y ajustándose a (3.18) conduce a que estos submódulos sean menores que en la aproximación 2*p*, reduciéndose el tamaño de los sumadores y, por tanto, incrementando su velocidad (hay que recordar que todas las sumas se realizan en paralelo, con lo que disminuye el tiempo total necesario para realizar la suma de los índices). La tabla 3.5 recoge una comparación entre ambas propuestas para ambas aproximaciones, mostrando el tamaño de las memorias de la figura

3.12, así como la longitud en bits de los sumadores requeridos. De la tabla 3.5 se deduce que:

a) el tamaño total de las memorias de índices (ROM1 y ROM2 en la figura 3.12) es siempre al menos un 10% inferior en la aproximación isomórfica, aunque en algunos casos el ahorro puede llegar al 50%;

b) el tamaño de la memoria inversa ROM3 se reduce, con un ahorro entre el 50% y el 75% (la única excepción es $p=103$);

c) aunque el hardware necesario para los sumadores, así como el retardo asociado a los mismos, depende de los submódulos elegidos, se puede concluir que el hardware requerido es siempre menor en la aproximación isomórfica que en la aproximación $2p$, lo que generalmente conduce a un menor retardo.

TABLA 3.5

p	Submódulos		ROM de índices (bits)		Tamaño sumadores (bits)		ROM inversa (bits)	
	aprox. $2p$	aprox. isomórfica	aprox. $2p$	aprox. isomórfica	aprox. $2p$	aprox. isomórfica	aprox. $2p$	aprox. isomórfica
19	6×7	2×9	384	320	3, 3	1, 4	320	160
31	7×9	2×3×5	448	384	3, 4	1, 2, 3	640	320
37	8×11	4×9	896	768	3, 4	2, 4	768	384
61	11×12	3×4×5	1024	896	4, 4	2, 2, 3	1536	768
67	11×13	2×3×11	2048	1792	4, 4	1, 2, 4	1792	896
79	13×14	2×3×13	2048	1792	4, 4	1, 2, 4	1792	896
89	13×14	8×11	2048	1792	4, 4	3, 4	1792	896
103	14×15	2×3×17	2048	2048	4, 4	1, 2, 5	1792	1792
113	15×16	7×16	2048	1792	4, 4	3, 4	1792	896
127	15×17	2×7×9	2304	2048	4, 5	1, 3, 4	3584	1792

Para ilustrar lo dicho, sea p=31, de modo que p-1 puede expresarse como 2×3×5, 3×10, 5×6 ó 2×15. Si se designa con $|m_j|$ el número de bits requeridos para operar con el submódulo m_j, tenemos que $|2|+|3|+|5|=6$, $|3|+|10|=6$, $|5|+|6|=6$ y $|2|+|15|=5$, así que el conjunto de submódulos {2,15} proporciona el menor tamaño de las memorias, en concreto, 2^5×5 bits para las tres memorias de la figura 3.12, al tiempo que se requieren sumadores de 1 y 4 bits. Sin embargo, {2, 3, 5} y {5, 6} dan el tamaño mínimo para los sumadores de cada submódulo; en concreto, {2, 3, 5} requiere ROM1 y ROM2 de 2^5×6 bits, y ROM3 de 2^6×5, mientras que los sumadores son de 1, 2 y 3 bits, respectivamente. Para {5, 6}, las memorias son del mismo tamaño, pero se necesitan dos sumadores de 3 bits.

En caso de que el criterio decisivo sea la velocidad, {2, 3, 5} ha de ser el conjunto de submódulos elegidos, ya que requiere la misma cantidad de memoria que {5, 6}, pero los sumadores son de menor tamaño, más simples y rápidos. Por contra, si el criterio dominante es el ahorro de memoria, {2, 15} han de ser los submódulos escogidos, ya que la cantidad de memoria requerida es menor, concretamente, 480 frente a 704 bits de memoria, un ahorro del 32%, pero asumiendo una penalización en la velocidad de operación, ya que se necesita un sumador de 4 bits (téngase en cuenta que la suma en módulo 2 se lleva a cabo con una simple puerta EXOR).

Tal y como se vio para la aproximación 2p, la figura 3.14 recoge el proceso de operación para un multiplicador módulo 29 construido con la aproximación isomórfica, con g=2 y submódulos {4, 7}, que satisfacen obviamente la condición (3.18) y son primos entre sí, tal y como se requiere para aplicar el teorema 3.3; del mismo modo que se hacía con la aproximación 2p, es posible detectar la multiplicación por cero, ya que (3.18) y el hecho de que los submódulos sean primos entre sí siempre asegura que algunos de éstos no es potencia de dos [GAR97a, GAR98c], tal y como se verá en el capítulo IV. Así, se puede asignar el valor 2^n-1=11...11_b como índice de cero para aquellos submódulos que no sean potencia de dos, como muestra la figura 3.14; posteriormente se verá la importancia de este hecho. La figura 3.14 también recoge el proceso de la multiplicación módulo 29 de x=5 e y=12, con $|xy|_{29}$=2. Los índices son, respectivamente, i(22)=(2,1) e i(7)=(3,0) en el sistema de residuos definido por los submódulos elegidos, {4, 7}; los

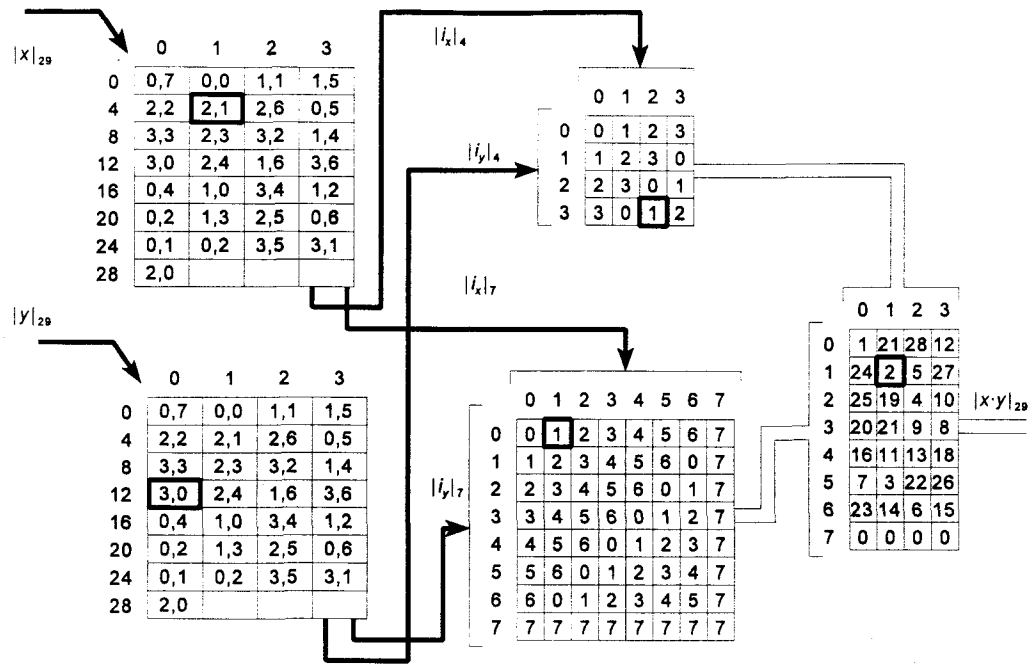


Figura 3.14. Multiplicador módulo 29 por cálculo de índices (aproximación isomórfica).

valores que entran en juego en el proceso de multiplicación en cada etapa aparecen recuadrados en la figura 3.14.

3.2.6. Multiplicadores de cuerpos de Galois en el SRNS

Ya se vio en la sección 2.1 que el SRNS permite representar números negativos haciendo uso de la representación en residuos, con una estructura muy similar al Sistema Numérico de Residuos convencional. La ubicación simétrica de los residuos en los diferentes módulos respecto de cero (recuérdese que el residuo x_i de X respecto del submódulo m_i está en el conjunto $\{-(m_i-1)/2,..., 0,..., (m_i-1)/2\}$) puede emplearse para mejorar el diseño de los multiplicadores en aritmética de residuos que hacen uso de las propiedades de los cuerpos de Galois; esto se debe a que se puede reducir el tamaño de las memorias a la mitad. De manera similar a los multiplicadores RNS basados en cuerpos de Galois, que se han llamado multiplicadores por cálculo de índices, los multiplicadores SRNS basados en cuerpos de Galois hacen uso de un elemento pseudoprimitivo, g, para la generación de los índices.

Definición 3.1: *sean las potencias de un entero* g *en módulo* p*:*

$$\left|g^{0}\right|_{p},\ \left|g^{1}\right|_{p},\ \ldots,\ \left|g^{(p-3)/2}\right|_{p}$$

Si sus magnitudes incluyen todos los valores 1, 2, ..., (p-1)/2, *entonces* g *recibe el nombre de* **raíz pseudoprimitiva** *en módulo* p.

Se puede demostrar que un elemento primitivo de GF(p) es también raíz pseudoprimitiva. Un entero X en el conjunto:

$$\left\{-\frac{p-1}{2},\ \ldots,\ 0,\ \ldots,\ \frac{p-2}{2}\right\}$$

bajo módulo p se representa en signo y magnitud como (s_x, x), donde s_x denota el bit de signo (0 para los números positivos y 1 para los negativos) y x indica la magnitud del número. La raíz pseudoprimitiva g permite representar también X como:

$$X = v_x \left|g^{i_x}\right|_p \qquad i_x \in \left\{0,\ 1,\ \ldots,\ \frac{p-3}{2}\right\} \qquad v_x \in \{0,\ 1\} \tag{3.19}$$

donde el nuevo bit de signo v_x no tiene el mismo significado que s_x. La tabla 3.6 recoge los valores de v_x e i_x para el SRNS módulo 29, con raíz pseudoprimitiva g=2:

TABLA 3.6

X	1	2	3	4	5	6	7	8	9	10	11	12	13	14
v_x	0	0	0	0	1	0	0	0	0	1	1	0	1	0
i_x	0	1	5	2	8	6	12	3	10	9	11	7	4	13
X	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10	-11	-12	-13	-14
v_x	1	1	1	1	0	1	1	1	1	0	0	1	0	1
i_x	0	1	5	2	8	6	12	3	10	9	11	7	4	13

Es fácil deducir que la tabla presenta simetría para los valores de i_x respecto de cero, por lo que es posible reducir el tamaño de las memorias de índices a la mitad, sin más qùe

considerar la magnitud del operando y generar v_x por separado. La forma de calcular v_x e i_x es intuitiva; si 1 en el RNS módulo p corresponde a 1 en el SRNS equivalente y p-1 corresponde a -1, y dado que g es elemento primitivo del RNS y raíz pseudoprimitiva del SRNS, se puede ver que:

$$i_{x_{SRNS}} = \left| i_{x_{RNS}} \right|_{\frac{p-1}{2}} \qquad v_x = \begin{cases} 0 & \text{si } i_{x_{RNS}} < \dfrac{p-1}{2} \\ 1 & \text{en otro caso} \end{cases} \tag{3.20}$$

El producto $z=|xy|_p$ usando la raíz pseudoprimitiva g se define como sigue:

$$|xy|_p = \begin{cases} v_{xy} \left| g^{i_x + i_y} \right|_p & \text{si } i_x + i_y < \dfrac{p-1}{2} \\ \bar{v}_{xy} \left| g^{i_x + i_y - \frac{p-1}{2}} \right|_p & \text{en otro caso} \end{cases} \tag{3.21}$$

donde $v_{xy} = v_x \oplus v_y$ De este modo, la figura 3.15 muestra el que sería el esquema del multiplicador en el SRNS, siendo:

$$l = \left\lceil \log_2 \frac{p+1}{2} \right\rceil \qquad k = \left\lceil \log_2 \frac{p-1}{2} \right\rceil \tag{3.22}$$

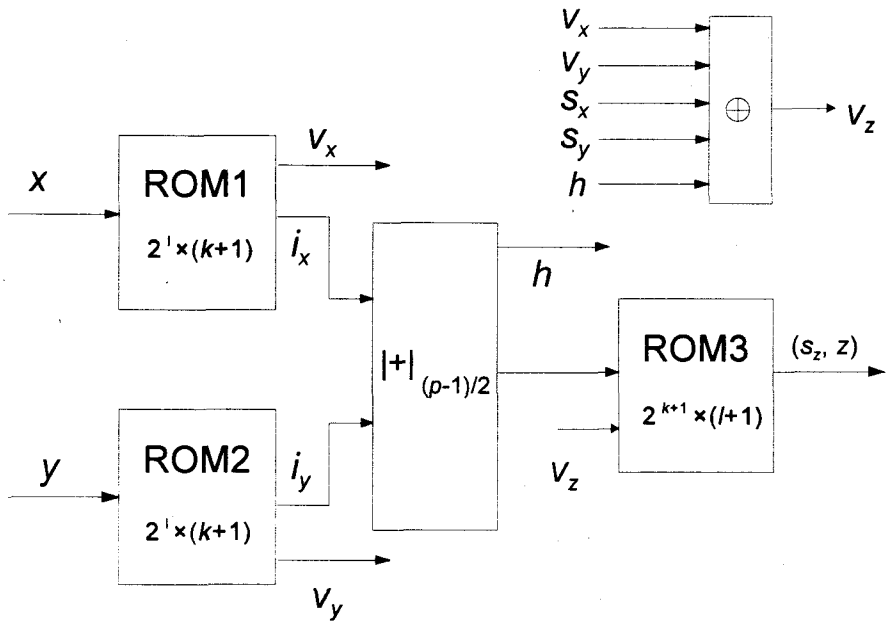


Figura 3.15. Multiplicador de cuerpos de Galois en el SRNS.

En este multiplicador, x, y y z se dan en signo y magnitud, de forma que ROM1 y ROM2 generan i_x e i_y a partir de las magnitudes de (s_x, x) y (s_y, y); dada la simetría que se veía antes, estas memorias tienen la mitad del tamaño que las equivalentes para un multiplicador RNS (figura 3.12); al mismo tiempo, el bit v se almacena sólo para los valores positivos, y en caso de que el operando sea negativo, sólo hay que invertirlo. Por su parte, el sumador genera un bit adicional h de la forma:

$$h = \begin{cases} 0 & \text{si } i_x + i_y < \dfrac{p-1}{2} \\ 1 & \text{en otro caso} \end{cases} \qquad \textbf{(3.23)}$$

que no es más que el indicador de desbordamiento de la suma módulo $(p\text{-}1)/2$. El bit v_z se genera como:

$$v_z = s_x \oplus s_y \oplus v_x \oplus v_y \oplus h \qquad \textbf{(3.24)}$$

que junto a la salida del sumador actúa como dirección de ROM3 para generar (s_z, z).

Las ventajas del uso del SRNS se centran en:

a) Dado que sólo hay que almacenar el índice de los operandos para los valores positivos (en la tabla 3.6 se observa que existe simetría), el tamaño de las memorias ROM1 y ROM2 es la mitad del tamaño de las memorias equivalentes de la figura 3.12, siendo sólo necesario añadir un bit adicional (el bit v) a cada palabra; la ROM3, sin embargo, es del mismo tamaño en ambos esquemas, ya que es necesario incluir números negativos y positivos.

b) La suma de los índices se realiza sobre un módulo menor que en el caso del multiplicador de la sección anterior, con lo que en principio puede realizarse a mayor velocidad.

El principal inconveniente (dado que la necesidad de generar el bit de signo no podría considerarse como tal, a la vista de su sencillez) se cifra en el hecho de que la aplicación de las aproximaciones anteriores (aproximación $2p$ y aproximación isomórfica

o técnica de mapeo isomórfico submodular) no es inmediata, con lo que estas técnicas pueden hacer que el esquema basado en el RNS convencional alcance una mayor velocidad al ser aplicadas, aunque a costa de mantener un tamaño sensiblemente mayor de las memorias que contienen los índices.

3.3. DIVISIÓN DE ENTEROS EN EL SISTEMA DE RESIDUOS

A continuación se va a mostrar uno de los algoritmos más recientes para la división en el RNS, a fin de ilustrar la complejidad de esta operación frente a la suma y la multiplicación. Hitz y Kaltofen [HIT95] proponen el uso de un sistema de residuos *extendido*; sean $m'_1,\ldots, m'_{2n}$ enteros primos entre sí tales que:

$$1 < m'_1 < m'_2 < \cdots < m'_{2n-1} < m'_{2n}$$

con n entero y $n \geq 1$. Estos módulos se agrupan en dos vectores de tamaño n:

$$m_1 = m'_1,\ m_2 = m'_3,\ \ldots,\ m_n = m'_{2n-1}$$

$$m_{n+1} = m'_2,\ m_{n+2} = m'_4,\ \ldots,\ m_{2n} = m'_{2n}$$

El sistema definido por estos módulos es lo que se denomina un sistema numérico de residuos extendido, con el RNS definido por los n primeros m_i como RNS *base*, y el RNS correspondiente al resto de módulos como RNS *extensión*. Los rangos respectivos para la base y la extensión son:

$$M_b = \prod_{i=1}^{n} m_i \qquad M_e = \prod_{i=n+1}^{2n} m_i$$

de forma que el rango total para el sistema extendido es el producto de éstos. Un entero X, $0 \leq X \leq M_b M_e$, con residuos $x_i = X \bmod m_i$, se representa en el RNS extendido como:

$$\left[x_1, \ldots, x_n;\ x_{n+1}, \ldots, x_{2n}\right]$$

De todo esto se pueden extraer las siguientes conclusiones:

a) M_b y M_e son primos entre sí, con $M_b<M_e$, por lo que existe M_b^{-1} mod M_e; además, la parte base de la representación en el RNS extendido es el residuo módulo M_b;

b) para valores de n suficientemente grandes y módulos de magnitud similar, la diferencia entre M_b y M_e es pequeña; en caso contrario, se pueden alterar los módulos para reducirla sin cambiar la propiedad anterior;

c) la multiplicación de dos números en el rango M_b llevada a cabo en el RNS extendido nunca producirá desbordamiento, ya que $M_b^2<M_bM_e$, y el resultado puede llevarse al rango original con un escalado por M_b.

El algoritmo para dividir X por Y, con $0\leq X<M_b$ e $0\leq Y<M_b$, calcula $\lfloor X/Y \rfloor$ y X mod Y:

```
begin
    Q ← ⌊X*RECIPROCO(Y)/M⌋
    R ← X - Q*Y
    if R<Y then begin
                    ⌊X/Y⌋ ← Q
                    X mod Y ← R
                end
           else begin
                    ⌊X/Y⌋ ← Q+1
                    X mod Y ← R - Y
                end
end
```

donde la función `RECIPROCO(Y)` devuelve el inverso de Y con respecto a M, $\lfloor M/Y \rfloor$; esta función hace uso de la iteración de Newton [LIP81] para realizar el cálculo:

$$Z_{i+1} = Z_i - \frac{f(Z_i)}{f'(Z_i)} \qquad \textbf{(3.25)}$$

donde:

$$f(Z_i) = \frac{M}{Z_i} - Y \qquad f'(Z_i) = -\frac{M}{Z_i^2} \tag{3.26}$$

y dado que en el RNS sólo se dispone de división entera, la recursión empleada es:

$$Z_{i+1} = \left\lfloor \frac{Z_i(2M - YZ_i)}{M} \right\rfloor = 2Z_i - \left\lfloor \frac{YZ_i^2}{M} \right\rfloor \tag{3.27}$$

La función `RECIPROCO(Y)`, con $1 \le Y \le M$, es:

```
begin
    Z1 ← 0
    Z2 ← 2
    while Z1≠Z2 do begin
                       Z1←Z2
                       Z2←⌊Z1*(2M-Y*Z1)/M⌋
                   end
    if M-Y*Z2<Y then ⌊M/Y⌋ ← Z2
                else ⌊M/Y⌋ ← Z2+1
end
```

La necesidad del último paso del algoritmo queda ilustrada por el siguiente ejemplo: con M=10 e Y=3, tenemos que $\lfloor M/Y \rfloor$=3, pero Z_2=Z_1=2, por lo que la corrección proporciona el resultado correcto. Se puede demostrar que estos algoritmos son correctos y proporcionan los resultados deseados.

Destaca la complejidad que implican operaciones como la división en el Sistema Numérico de Residuos, lo que limita las aplicaciones del mismo, tal y como se ha comentado en ocasiones anteriores; en concreto, aplicando lo visto en esta sección a lo expuesto en otros capítulos, se observa que la división implica la realización de operaciones de suma y multiplicación durante un número desconocido *a priori* de iteraciones, tal y como muestra la función `RECIPROCO`. Esto supone una enorme

penalización en el caso de ser necesaria una operación de división durante el uso del sistema de residuos. Por contra, este hecho ayuda a centrar el interés de los investigadores en la mejora del hardware asociado a las operaciones de suma y multiplicación y en la búsqueda de algoritmos más sencillos para operaciones como el escalado; el primer objetivo ayuda a mejorar las prestaciones alcanzadas en las aplicaciones clásicas del RNS (filtrado digital, transformadas, etc., y, en general, procesamiento digital de señales) en las que principalmente intervienen las operaciones de suma y multiplicación; pero esto contribuye también a la mejora en las operaciones de división y escalado, de manera que algoritmos sencillos y eficientes unidos a realizaciones cada vez mejores de las operaciones básicas permiten ir mejorando día a día la realización de operaciones que incluso se pretenden evitar en lo posible.

3.4. CONCLUSIÓN

En este capítulo se han expuesto diversas propuestas sobre el diseño de sumadores y multiplicadores para el Sistema Numérico de Residuos, orientadas a diferentes aplicaciones, a diferentes tamaños de módulos, etc. Concretamente, se ha ido desde diseños de carácter general para módulos grandes y muy cercanos a otro tipo de esquemas aritméticos convencionales, a propuestas de carácter muy específico, que tratan de sacar el máximo partido no sólo de las propiedades del Sistema Numérico de Residuos, sino de las posibilidades que ofrece el álgebra a través del estudio de los cuerpos finitos.

La principal conclusión que se debe extraer de este capítulo es la confrontación entre las potencialidades del RNS y sus limitaciones. Efectivamente, se acaba de ver en la sección 3.2.3 propuestas que simplifican de manera sorprendente una operación básica como la multiplicación; esto hace que el sistema de residuos sea una herramienta no despreciable a la hora de realizar aplicaciones que involucren una gran cantidad de productos. Sin embargo, la sección 3.3 ha mostrado de manera dramática la principal limitación del RNS, la división, ya que las mismas características de este sistema que facilitan suma y producto hacen que las dificultades intrínsecas de la operación de división se vean magnificadas.

CAPÍTULO IV

MULTIPLICADORES PARA MÓDULOS PRIMOS CON CAUCE SEGMENTADO

En este capítulo se presentan los primeros resultados del trabajo incluido en esta memoria; en concreto, se aplicarán algunos de los conceptos vistos en los capítulos II y III para el diseño de un multiplicador módulo un número primo con cauce segmentado. Como ejemplo, se mostrarán los resultados para un multiplicador módulo p=127.

4.1. INTRODUCCIÓN

En el capítulo III se mostró la multiplicación según la técnica de mapeo isomórfico submodular o aproximación isomórfica, que constituye una aproximación sencilla y eficiente para el cálculo de la multiplicación módulo un número primo a través de las propiedades de los cuerpos finitos. En este sentido, incluso ha sido desarrollado el GERNS (Galois Enhanced Residue Number System) [ZEL91, SMI95], usando un sistema de residuos definido por módulos primos y realizando una transformación desde el dominio de los residuos al dominio de los correspondientes índices; si bien esta alternativa elimina el cálculo de índices para la multiplicación, al desarrollarse todo el procesamiento en este dominio la suma de residuos se transforma en una suma logarítmica de índices. Esta suma logarítmica [ZEL91] implica una complejidad similar a la multiplicación por cálculo de

índices cuando el procesamiento se realiza sobre el dominio de los residuos, con lo que el uso del GERNS sólo está justificado en situaciones donde la operación predominante sea la multiplicación.

Como primera contribución de la investigación recogida en esta memoria, en este capítulo se presentan algunas modificaciones y mejoras a los multiplicadores módulo un número primo por mapeo isomórfico submodular. Si bien esta estructura fue propuesta hace dos décadas, la necesidad de tres tablas para el cálculo de índices y la regeneración del resultado ha limitado su uso. Más aún, en la sección 3.2.3 se mostró la idea básica para incluir la multiplicación por cero en este esquema; sin embargo, las propuestas previas exigían la representación del índice de cero para cada uno de los submódulos; se mostrará que esto no es correcto y que sólo se requiere la representación ficticia de cero para uno de los submódulos.

Finalmente, se extenderán las ideas empleadas para la inclusión de la multiplicación por cero en el mapeo isomórfico submodular para la implementación de un sencillo esquema de detección de errores en el multiplicador con un coste prácticamente nulo.

4.2. SEGMENTACIÓN DE CAUCE

El esquema de la figura 3.12, dada su sencillez, es susceptible de ser implementado con una estructura segmentada; de este modo, dicho esquema se transformaría en el que se muestra la figura 4.1. En ésta, CLK es la señal de reloj del sistema, y CS1, CS2 y CS3 son las hipotéticas señales de selección de chip (chip-select) para las memorias o dispositivos de las tres etapas, respectivamente. De esta forma, se puede disponer en cada pulso de reloj de un dato de salida, tres pulsos después de que se presentaran los datos en la entrada del multiplicador.

4.2.1. Multiplicación por cero

La implementación de la figura 4.1 presenta un inconveniente, que se refiere a la presencia de un valor cero en una de las entradas; si la realización no estuviera segmentada, una puerta OR o NOR de k entradas, siendo k el número de bits necesario para representar valores módulo p, bastaría para detectar la combinación $00..00_b$ en cada una de las entradas; de este modo, se podría dar como salida $00..00_b$ sin necesidad de procesar o encaminar los datos de entrada a través del multiplicador, reduciendo considerablemente el tiempo de operación si se multiplica por cero. En el caso de que se segmente la estructura del multiplicador, no se puede implementar este mecanismo, ya que los resultados deben ir apareciendo en la salida en el mismo orden en que los datos de entrada correspondientes llegaron al circuito, a fin de sincronizar correctamente entradas y salidas. Si la multiplicación por cero se realizara como hemos descrito, detectando entrada cero (un valor cero en una de las entradas) con datos en proceso en cada una de las etapas, interrumpiría el proceso, poniendo en la salida el valor $00..00_b$, lo que alteraría el orden de la secuencia de datos de salida con respecto del orden de la de entrada.

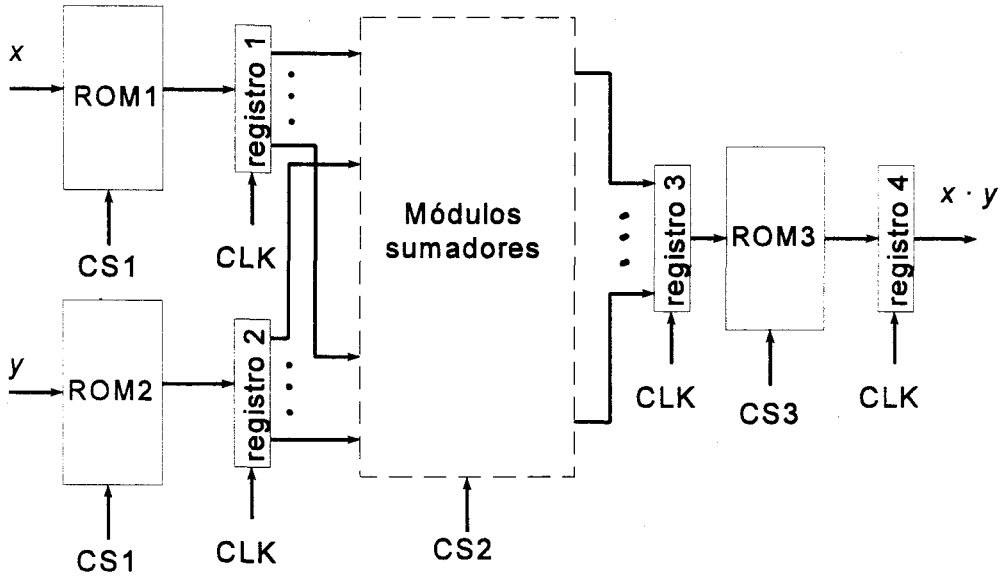


Figura 4.1. Esquema simplificado del multiplicador módulo p con cauce segmentado.

Se hace, pues, necesario implementar algún mecanismo que permita llevar a cabo la multiplicación por cero [GAR97a] sin afectar la segmentación de cauce del multiplicador; la clave está una vez más en la sección 3.2.3, concretamente en el mecanismo implementado en las figuras 3.13 y 3.14; de este modo, si se asigna para todos los submódulos no potencia de dos el valor $2^{k_i} - 1$ (donde k_i es el número de bits necesario para operar con el submódulo m_i) como residuo del índice de cero para dichos submódulos en la entrada 0 de las memorias ROM1 y ROM2, se pueden implementar los sumadores correspondientes contemplando que:

$$(|i_x|_{m_i} + |i_y|_{m_i}) \bmod m_i = \begin{cases} 2^{k_i} - 1 & \text{si } |i_x|_{m_i} = 2^{k_i} - 1 \\ 2^{k_i} - 1 & \text{si } |i_y|_{m_i} = 2^{k_i} - 1 \\ (|i_x|_{m_i} + |i_y|_{m_i}) \bmod m_i & \text{en cualquier otro caso} \end{cases} \quad \textbf{(4.1)}$$

Por tanto, si en una de las entradas de los sumadores se presenta el valor $2^{k_i} - 1$, que no será un residuo válido si el submódulo no es potencia de dos, la salida será $2^{k_i} - 1$, de forma que en la ROM3, la dirección formada sólo por unos en los bits correspondientes a los submódulos que no son potencia de dos debe contener como resultado $00..00_b$, es decir, el resultado de multiplicar por cero. De este modo [GAR97a, GAR98c], se observa que no es necesario que el índice de cero se adjudique a $11..11_b$ para todos los submódulos, en contra de lo que había sido afirmado por otros autores [JUL80] y se refleja en la figura 3.13 para la aproximación 2*p*; basta con considerar sólo los submódulos que no son de la forma 2^{k_i}, con k_i entero.

Esta idea que se acaba de exponer es muy interesante, ya que permite considerar un caso especial sin modificar la estructura de la figura 4.1, sin más que alterar el contenido de las memorias y modificar ligeramente los sumadores; además, estas posiciones de memoria que se ven alteradas no habrían de usarse si no se llevara a cabo esta forma de detección de cero en la entrada; éstas corresponden a combinaciones de entradas que no deberían de presentarse en ningún momento durante el funcionamiento normal del sistema, o que en el caso de hacerlo, como el valor 0 en una de las entradas, necesitarían un hardware especial para resolver la situación que plantean.

4.2.2. Detección de errores

La solución anterior para la multiplicación por cero puede extenderse para una mayor funcionalidad, ya que 0 no es el único valor especial que podemos encontrar en las entradas; en particular, y dado que la técnica de mapeo isomórfico submodular requiere que el módulo p sobre el que se realiza la multiplicación sea primo, existirán direcciones en las memorias ROM1 y ROM2 de la figura 4.1 que no correspondan a datos válidos módulo p. Dado que $2^{k_p}-1$ (k_p es el numero de bits necesario para representar datos módulo p) nunca será un valor válido módulo p, podemos usarlo como indicador de error o de entrada inválida y darlo como resultado en caso de que alguno de los valores de los operandos de entrada sea mayor o igual que p; esto es razonable ya que $2^{k_p}-1$ es un valor fácilmente detectable por parte del sistema que haga uso de los resultados proporcionados por el multiplicador propuesto.

De nuevo se presenta el problema de cómo detectar valores incorrectos y cómo llevar la combinación $2^{k_p}-1$ a la salida; puesto que pueden ser más de una las combinaciones incorrectas en las entradas, la detección no es tan sencilla como para 0; además, es necesario incluir este proceso en la estructura segmentada, al igual que para la multiplicación por cero. Con estas restricciones presentes, es inmediato decantarse por la misma solución que la implementada para la multiplicación por cero, pero teniendo en cuenta ciertas limitaciones. Concretamente, supóngase que existe algún submódulo m_i de los definidos por el teorema 3.3 para el que se satisface que $m_i \leq 2^{k_i}-2$, siendo k_i el número de bits necesarios para la representación en módulo m_i. Por tanto, no sólo $2^{k_i}-1$ es incorrecto módulo m_i, sino que $11..10_b=2^{k_i}-2$ tampoco es válido. De este modo, podemos asignar este valor como índice del error, al igual que $2^{k_i}-1$ es el índice de cero, y hacer que los bits correspondientes a los submódulos $m_i \leq 2^{k_i}-2$ en las memorias ROM1 y ROM2 tomen el valor $2^{k_i}-2$ para todas las entradas inválidas módulo p. Además, el correspondiente sumador módulo m_i debe contemplar que el valor $11..10_b=2^{k_i}-2$ en una de las entradas debe producir $2^{k_i}-2$ como salida, sea cual sea el valor de la otra entrada, ya que esto indica que uno de los multiplicandos es un número incorrecto y que el resultado de la multiplicación debe ser $2^{k_p}-1$ para indicar este error.

De este modo, para completar la detección del error en la entrada, la memoria ROM3 debe contener $2^{k_p} - 1$ en todas las palabras correspondientes a las direcciones en las que los bits correspondientes a $m_i \leq 2^{k_i} - 2$ presenten el valor $2^{k_i} - 2$.

Además, este mismo mecanismo también permite implementar cierta detección de errores en el cálculo sobre los submódulos $m_i \leq 2^{k_i} - 2$, sin más que hacer que la salida del sumador sea $2^{k_i} - 2$ no solamente cuando una entrada toma el valor $2^{k_i} - 2$, sino en todos los casos en que una o las dos entradas correspondan a valores no válidos módulo m_i. De la misma forma, en ROM3 todas las direcciones en que los bits correspondientes a $m_i \leq 2^{k_i} - 2$ no representen un valor correcto en módulo m_i deben producir como salida $2^{k_p} - 1$, valor que se ha tomado como indicativo del error. Así, la alteración de un dato correspondiente a m_i que lleve el residuo en curso a un valor incorrecto es detectada, provocando como resultado de la multiplicación $2^{k_p} - 1$.

En el caso en que $m_i = 2^{k_i} - 1$, sólo es posible implementar la detección de la multiplicación por cero, ya que $2^{k_i} - 2$ es un dato válido; en este caso, se puede limitar a otorgar el residuo $2^{k_i} - 1$ en las memorias ROM1 y ROM2 para la única combinación incorrecta, que es precisamente $2^{k_i} - 1$, con lo que un valor incorrecto en la entrada produciría cero como resultado de la multiplicación.

Para $m_i = 2^{k_i}$, no es posible implementar ninguno de estos mecanismos, por lo que al definir el contenido de las diferentes memorias, se deberá ignorar este submódulo (del teorema 3.3 es inmediato deducir que sólo unos de los submódulos puede ser potencia de dos) a la hora de implementar los mecanismos de detección de multiplicación por cero y de detección de errores.

Al definir el contenido de las memorias también habrá que decidir la condición que predomina; por tanto, si a ROM3 llega una dirección con $2^{k_i} - 1$ para todos los submódulos, excepto para uno en el que haya $2^{k_i} - 2$ u otra combinación incorrecta, se podrá dar como salida cero ó $2^{k_p} - 1$, según se quiera que prevalezca la multiplicación por cero o el error en el submódulo con $2^{k_i} - 2$. Parece lógico pensar que la condición de error ha de prevalecer sobre la multiplicación por cero, de forma que en la situación anterior la

salida debería ser $2^{k_p}-1$. De este modo, para aquellos submódulos que satisfagan $m_i \le 2^{k_i}-2$, la ecuación (4.1) se transforma en:

$$(|i_x|_{m_i}+|i_y|_{m_i}) \bmod m_i = \begin{cases} 2^{k_i}-1 & \text{si } |i_x|_{m_i}=2^{k_i}-1 \text{ e } |i_y|_{m_i}<m_i \\ 2^{k_i}-1 & \text{si } |i_y|_{m_i}=2^{k_i}-1 \text{ e } |i_x|_{m_i}<m_i \\ 2^{k_i}-1 & \text{si } |i_x|_{m_i}=|i_y|_{m_i}=2^{k_i}-1 \\ 2^{k_i}-2 & \text{si } m_i \le |i_x|_{m_i} \le 2^{k_i}-2 \\ 2^{k_i}-2 & \text{si } m_i \le |i_y|_{m_i} \le 2^{k_i}-2 \\ (|i_x|_{m_i}+|i_y|_{m_i}) \bmod m_i & \text{en cualquier otro caso} \end{cases} \quad \textbf{(4.2)}$$

Antes de seguir, se van a ilustrar estos dos mecanismos para el módulo propuesto, p=127, de forma que p-1=2×7×9; así, 2 no permite implementar ningún tipo de detección; 7 permite la detección de la multiplicación por cero y 9 es adecuado para llevar a cabo todo lo dicho; de este modo, y en primer lugar, ROM1 y ROM2 deben contener:

TABLA 4.1

Dirección	Contenido	
0000000	1 111 1111	se multiplica por cero
0000001	0 000 0000	
.	.	
.	.	
.	.	
1111111	1 111 1110	valor no válido

donde los bits de cada submódulo se colocan en el orden 2, 7 y 9, de izquierda a derecha; así, para $127=2^{k_p}-1=1111111_b$ el bit correspondiente al submódulo 2 es 1, para m_2=7 tenemos $111_b=2^{k_2}-1$, y para m_3=9, $1110_b=2^{k_3}-2$, valor que posibilita la detección de errores. Dado que en módulo 2 no se puede aplicar ningún tipo de detección, el sumador sigue siendo una puerta EXOR.

El sumador para el submódulo 7 puede sólo incluir la multiplicación por cero, por lo que la única situación especial corresponde a la ecuación (4.1), tal y como muestra la tabla 4.2:

TABLA 4.2

Sumando 1	Sumando 2	Resultado
×××	111	$111=2^{k_2}-1$
111	×××	$111=2^{k_2}-1$

Para el sumador módulo 9, la funcionalidad a incluir corresponde a la ecuación (4.2), tal y como recoge la tabla 4.3.

TABLA 4.3

Sumando 1	Sumando 2	Resultado	
××××	1001	$1110=2^{k_3}-2$	error
××××	1010	$1110=2^{k_3}-2$	error
××××	1011	$1110=2^{k_3}-2$	error
××××	1100	$1110=2^{k_3}-2$	error
××××	1101	$1110=2^{k_3}-2$	error
××××	1110	$1110=2^{k_3}-2$	error
$\geq m_3$	1111	$1110=2^{k_3}-2$	error
$< m_3$	1111	$1111=2^{k_3}-1$	mult. cero
1111	1111	$1111=2^{k_3}-1$	mult. cero

Por último, señalar que la tabla 4.3 no recoge el intercambio del sumando 1 por el sumando 2, que habría de producir resultados equivalentes.

Por último, la memoria ROM3 también debe recoger todo lo dicho, combinando los residuos en los diferentes submódulos para producir las salidas adecuadas; en concreto, la tabla 4.4 recoge el contenido de algunas de las direcciones de ROM3. La multiplicación

por cero sólo se considera si todos los bits correspondientes a los submódulos que no son potencia de dos son uno; esto es así ya que el valor cero en una de las entradas provoca que uno de los sumandos sea $2^{k_i} - 1$ para todos los submódulos que no son potencia de dos. Por tanto, de acuerdo a lo dicho, todos estos sumadores deberán dar como salida $2^{k_i} - 1$, salvo que se produzca un error o que el otro multiplicando sea un valor incorrecto (mayor o igual que p), con lo que para los submódulos con $m_i \leq 2^{k_i} - 2$, y según se ha señalado, la otra entrada del sumador es $2^{k_i} - 2$, que debe producir $2^{k_i} - 2$, que en ROM3 será indicativo también de error.

TABLA 4.4

Dirección	Contenido	
0 000 0000	0000001	valores correctos
.	.	valores correctos
.	.	valores correctos
.	.	valores correctos
× ××× 1001	$1111111=2^{k_p} - 1$	valor no válido (error)
× ××× 1010	$1111111=2^{k_p} - 1$	valor no válido (error)
× ××× 1011	$1111111=2^{k_p} - 1$	valor no válido (error)
× ××× 1100	$1111111=2^{k_p} - 1$	valor no válido (error)
× ××× 1101	$1111111=2^{k_p} - 1$	valor no válido (error)
× ××× 1110	$1111111=2^{k_p} - 1$	valor no válido (error)
× ××× 1111	$1111111=2^{k_p} - 1$	valor no válido (error)
× 111 ××××	$1111111=2^{k_p} - 1$	valor no válido (error)
× 111 1111	0000000	multiplicación por cero

4.3. DISEÑO DEL MULTIPLICADOR

Tras exponer en la sección 4.1 algunos de los aspectos relacionados con la segmentación de cauce de la estructura en la figura 4.1, vamos ahora a detallar la

estructura del multiplicador para la posterior implementación de un ejemplo práctico.

4.3.1. Control de la segmentación de cauce

En principio [GAR98c], la única dificultad para la segmentación de cauceradica en el control del cauce segmentado; puesto que un sistema como éste, un multiplicador, estará en principio destinado a integrarse en un sistema de mayor complejidad, se supondrá que el control sobre el multiplicador será ejercido por dicho sistema. En concreto, se impondrá que el inicio de la multiplicación, es decir, la presencia de un dato válido en las entradas del mismo, esté señalado por una señal DI (Data Input); de este modo, si esta señal tiene el valor 1 durante el flanco de subida de la señal de reloj, los datos de entrada deberán ser procesados en los siguientes pulsos de reloj.

Por otra parte, en la figura 4.1 aparecen las señales CS× para la selección o activación de las diferentes memorias y sumadores; así, la lógica de control del multiplicador habrá de generar éstas de manera adecuada. Además, si se usa la señal DI para indicar la presencia de un dato válido en las entradas, es lógico pensar en la

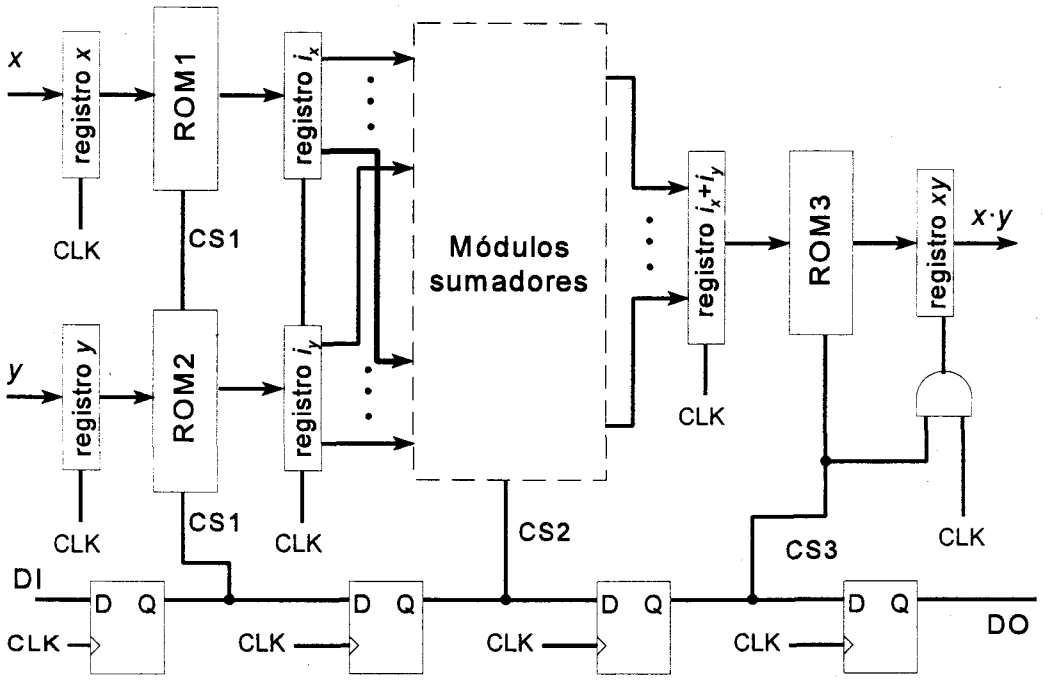


Figura 4.2. Señales de control del multiplicador con segmentación de cauce.

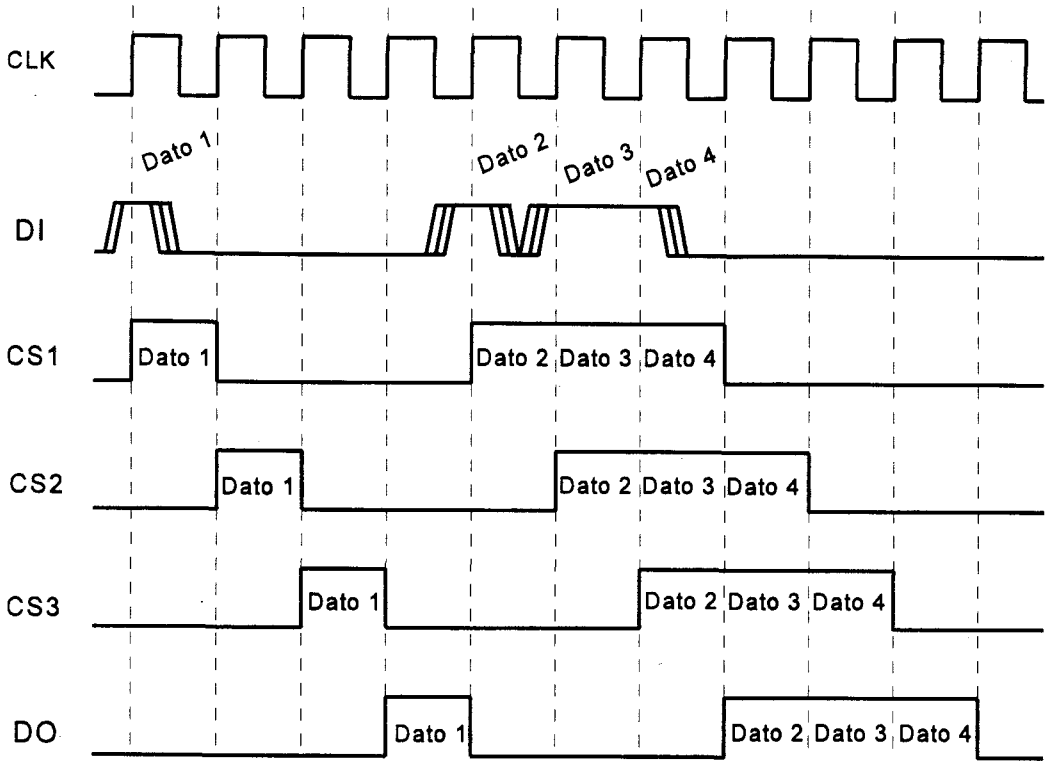


Figura 4.3. Cronograma simplificado del multiplicador.

generación de una señal DO (Data Output) que indique que en la salida del multiplicador está presente un resultado válido. Todo esto parecer complicar las funciones a incluir en la lógica de control. Sin embargo, la figura 4.2 muestra que esto no es así, de forma que es posible generar todas estas señales a partir de DI, simplificando enormemente el control del cauce segmentado del multiplicador. De este modo, una cadena de biestables permite llevar a cada elemento del multiplicador la señal CS× correspondiente a cada conjunto de datos a partir del correspondiente valor lógico 1 en DI. De igual manera, la generación de DO se reduce a llevar a la salida de la cadena de control la señal DI. Los registros situados entre las diferentes etapas del multiplicador permiten que en cada ciclo de reloj se obtenga un resultado.

La figura 4.3 muestra un cronograma simplificado del funcionamiento del multiplicador de la figura 4.2. En esta figura, las etiquetas "Dato ×" hacen referencia al momento en que se señala que el par de entradas x e y es válido, además del dato que se procesa en cada momento para las señales CS1, CS2 y CS3, o el resultado que valida DO. Como se puede ver en la figura 4.3, DI sólo ha de estar activa durante el flanco de subida

de la señal de reloj, CLK, de modo que el primer biestable en la cadena de control del cauce segmentado genera CS1, al tiempo que los datos de entrada quedan almacenados en los registros x e y de la figura 4.2. En el siguiente flanco de CLK, la salida de las tablas de índices, ROM1 y ROM2, se almacenan en los registros i_x e i_y, respectivamente, mientras que la cadena de biestables pasa a generar CS2 a partir del valor anterior de CS1. Por tanto, en el siguiente flanco del reloj, la suma de los índices se carga en el registro i_x+i_y y se genera CS3. Finalmente, un nuevo flanco en CLK activa la señal DO y carga en el registro xy el resultado de la multiplicación correspondiente. Por otra parte, si el multiplicador se integra en un sistema más complejo en el que el elemento que proporcione las entradas al multiplicador esté integrado en la segmentación de cauce, con un registro a la salida, es posible eliminar los registros de entrada x e y en la figura 4.2, ya que serían innecesarios y añadirían un ciclo de latencia al cauce total del sistema.

4.3.2. Implementación de los sumadores

En el apartado anterior se ha descrito la estructura básica del multiplicador con cauce segmentado; por otra parte, la sección 4.2 enumera las diferentes posibilidades para el multiplicador, tales como inclusión de la multiplicación por cero en el cauce o la detección de errores. Ninguna de estas funciones tiene repercusión sobre el cálculo de índices, ya que no implican más que modificar el contenido de las tablas de índices, ROM1 y ROM2, y el de la tabla que almacena el cálculo inverso, ROM3. Sin embargo, los sumadores han de ser convenientemente modificados para aquellos submódulos que no sean potencia de dos, tal y como indican las ecuaciones (4.1) y (4.2); de este modo, no es posible definir *a priori* qué estructuras son las más adecuadas para implementar dichos sumadores. En principio, las arquitecturas propuestas en el capítulo III habrían de ser modificadas para incluir las nuevas funciones; por contra, la implementación de los sumadores con tablas de consulta permite implementar estas y otras funciones sin coste adicional. Si bien la alternativa basada en tablas puede conducir a una cantidad excesiva de memoria (recuérdese que serían necesarias tablas de dos entradas), el hecho de usar la aproximación isomórfica submodular (teorema 3.3) hace que, en principio, los submódulos a emplear requieran un menor número de bits que p, con lo que es factible la

implementación de los sumadores para los submódulos satisfaciendo $m_i \leq 2^{k_i} - 2$ y $m_i \leq 2^{k_i} - 1$ a partir de tablas de memoria. De este modo, en lo que sigue, esta será la alternativa elegida.

4.4. IMPLEMENTACIÓN PARA p=127

Para finalizar este capítulo, esta sección describe el diseño concreto para un multiplicador módulo p=127 usando la técnica de mapeo isomórfico submodular; la elección de este valor de p no es arbitraria y en capítulos sucesivos se mostrará que módulos de cinco a siete bits son los más útiles para el desarrollo de aplicaciones basadas en el RNS con la tecnología disponible en estos momentos. Además, es obvio que p=127 satisface las condiciones del teorema 3.3, y en especial (3.18).

Tal y como se ha reflejado en la sección 4.3.2, los sumadores para los submódulos con $m_i \leq 2^{k_i} - 2$ y $m_i \leq 2^{k_i} - 1$ serán implementados directamente sobre tablas de consulta; de este modo, se ha de realizar en primer lugar la elección óptima de acuerdo con los criterios expuestos en el apartado 3.2.3. Puesto que se ha elegido la implementación sobre tablas de consulta para los sumadores, el criterio predominante será el del uso mínimo de memoria, ya que la velocidad del multiplicador vendrá determinada por la latencia de las memorias empleadas. De este modo, son varias las descomposiciones posibles de p=127, mientras que podemos definir el parámetro n como:

$$n = \sum_{i=1}^{r} k_i \equiv \sum_{i=1}^{r} \lceil \log_2 (m_i - 1) \rceil \qquad \textbf{(4.2)}$$

es decir, el número de bits necesarios para la representación en el sistema de submódulos de los índices correspondientes a las entradas y resultados. La tabla 4.5 muestra los valores de n y la memoria requerida en bits para las posibles descomposiciones de p=127 según (3.18). Por tanto, la elección de los submódulos ha de ser **{2, 7, 9}**, a pesar de que el valor de n es mayor que para otros conjuntos de submódulos; la diferencia viene marcada por el hecho de que las elecciones en que hay algún submódulo de valor elevado, la cantidad

de memoria requerida para la suma en dicho submódulo se dispara. Finalmente, la estructura para un multiplicador módulo 127 queda reflejada en la figura 4.4.

TABLA 4.5

$\{m_i\}$	n	Memoria requerida (bits)
2, 7, 9	8	5056
2, 63	7	27264
9, 14	8	5888
7, 18	8	8960
126	7	117376

Como ya se ha comentado anteriormente (capítulo I, sección 3.1.3), dispositivos programables como FPGAs o CPLDs están revolucionando el mundo del diseño digital con sistemas cada vez más complejos. De este modo, el diseño aquí propuesto puede ser albergado sin dificultad por cualquiera de los dispositivos programables modernos; además, la configuración de los bloques lógicos básicos en la mayoría de fabricantes es

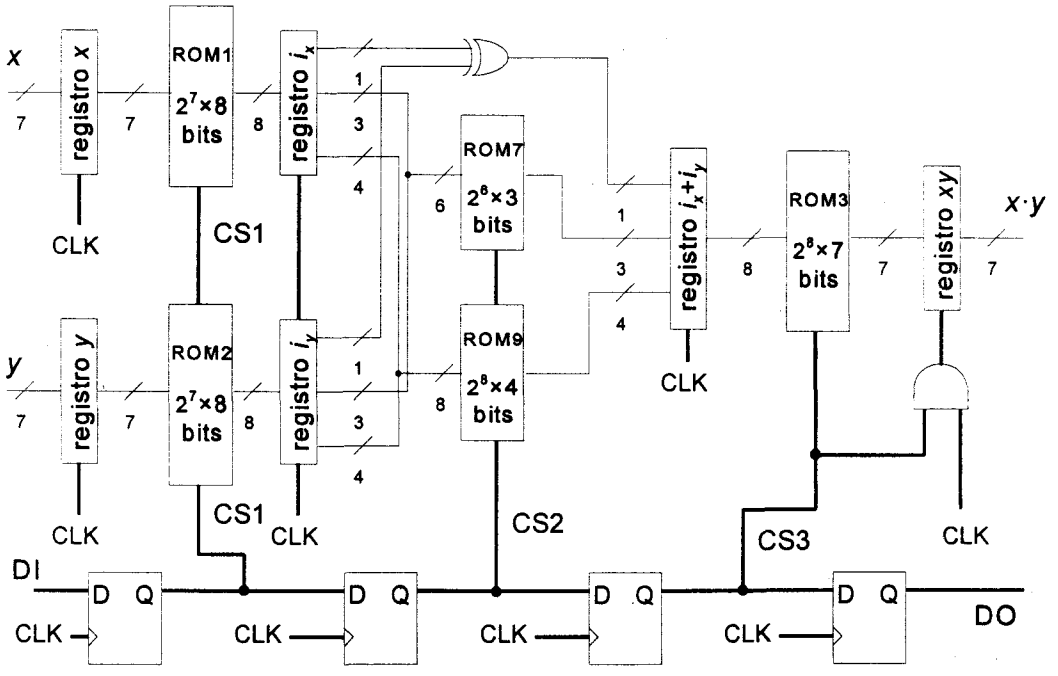


Figura 4.4. Multiplicador para p=127.

tal que la construcción de circuitos con segmentación de cauce no supone un coste adicional, ya que cada bloque lógico incluye un biestable en su salida, con lo que la implementación de los registros necesarios para la segmentación de cauce no hace uso de ningún tipo de recurso adicional.

Por otra parte, la presencia de memorias usadas como tablas en este multiplicador tampoco supone un problema para las más modernas familias de dispositivos programables, como los CPLDs FLEX10K de Altera (v. apéndice B), ya que no es necesario emplear un elevado número de bloques lógicos para implementar una pequeña tabla, al estar parte del dispositivo formado por recursos específicos que pueden funcionar como RAM o ROM. En concreto, la familia FLEX10K de Altera dispone de tablas de $2^8 \times 8$ bits, que pueden albergar perfectamente todas las tablas requeridas en el diseño de la figura 4.4. Las cinco tablas que se utilizan en la figura 4.4 se encuentran disponibles en los dispositivos de tamaño medio de la mencionada familia. Lo que es más, los chips de mayor capacidad de estas familias permiten incluir en la actualidad, gracias a la presencia de estas tablas, aplicaciones completas de procesamiento digital de señales, tal y como veremos en el capítulo VI.

UNIVERSIDAD DE GRANADA
13 MAYO 1999
COMISION DE DOCTORADO

4.5. CONCLUSIÓN

Se ha mostrado en este cuarto capítulo cómo plasmar en la práctica algunas de las ideas expuestas en el capítulo III, con las diferentes propuestas para la construcción de un multiplicador módulo p con la técnica de mapeo isomórfico submodular, al tiempo que se han introducido mejoras sustanciales a las propuestas existentes en la literatura; se pueden resumir las principales aportaciones del capítulo en los siguientes puntos:

- Se ha mostrado que no es necesario para la técnica de mapeo isomórfico submodular que sea posible asignar un índice ficticio a cero para cada submódulo a fin de incluir la multiplicación por cero en un cauce segmentado, en contra de lo afirmado por otros autores; más aún, se ha mostrado que sólo se requiere que uno de los submódulos satisfaga $m_i \leq 2^{k_i} - 1$ para que esto sea posible.

- Se ha introducido un esquema de detección de errores en el multiplicador basado en las mismas ideas que el proceso de multiplicación por cero; en concreto, se ha mostrado que solamente es necesario que uno de los submódulos satisfaga $m_i \leq 2^{k_i} - 2$ para que este mecanismo sea posible.

- Dado que las operaciones de cálculo de índices y cálculo inverso se realizan a partir de tablas, el coste que implican las funciones anteriores es nulo en lo que se refiere a estas dos operaciones; además, cuando los sumadores para los submódulos que implementan la multiplicación por cero, la detección de errores o ambas funciones se construyen sobre una tabla directa de consulta, el coste añadido por éstas es nulo; de este modo, estos mecanismos son completamente compatibles con la segmentación de cauce del multiplicador.

- El multiplicador propuesto puede ser construido sobre dispositivos programables, más aún cuando las últimas familias de CPLDs facilitan la construcción de arquitecturas con cauce segmentado e incluyen recursos específicos para la implementación de tablas de consulta en ROM.

Finalmente, en el capítulo V se mostrará una aplicación de la arquitectura detallada en este capítulo con el desarrollo de nuevas propuestas para el escalado en el RNS.

CAPÍTULO V

ESCALADO EN EL RNS

Este capítulo recoge las propuestas resultantes de la investigación realizada en el campo del escalado en el RNS; en concreto, se recoge una modificación de algunos de los algoritmos previos para una implementación más eficiente en términos de relación entre velocidad y coste, haciendo uso de los multiplicadores propuestos en el capítulo IV. Por otra parte, se presenta un nuevo método basado completamente en el uso de tablas de consulta con un tiempo de cálculo equivalente a dos ciclos de acceso a memoria; además, se mostrará una implementación VLSI de este algoritmo para un RNS de tres módulos.

5.1. INTRODUCCIÓN

Como ya se comentó en el capítulo II, el escalado es una operación delicada en el RNS. Esto es así ya que el escalado es una forma especial de división, y como hemos mostrado en la sección 3.3, la complejidad que implica la división en el RNS hace de ella una operación irrealizable en la mayoría de supuestos prácticos. Sin embargo, el hecho de que el divisor sea una constante en el escalado simplifica la operación y la hace factible. Más aún, existen aplicaciones en las que el ritmo de crecimiento de los datos procesados hace obligatorio el escalado a fin de obtener resultados correctos (v. apartado 2.2.2). Si bien es posible el definir un RNS con un rango dinámico que cubra el crecimiento máximo de los datos, esta solución puede resultar menos ventajosa en términos de hardware que

una estrategia adecuada de escalado.

Aunque existen varios algoritmos de escalado [JUL78, GRI89, SU90], aquéllos basados en cálculos iterativos del tipo MRC [JUL78, GAR98d] han sido los más empleados para la implementación directa del escalado. De este modo, se presentarán dos nuevos esquemas para el escalado en el RNS basados en este tipo de algoritmos y aplicables en diversas situaciones.

5.2. ALGORITMOS CLÁSICOS PARA EL ESCALADO EN EL RNS

Antes de presentar las propuestas resultantes de la investigación realizada, se van a presentar en esta sección las ecuaciones fundamentales del escalado y las implementaciones clásicas de esta operación.

5.2.1. Ecuaciones fundamentales del escalado

Dado que la división no es una operación cerrada en Z_m, el escalado no puede calcularse de manera exacta; de este modo, si K es el factor o constante de escalado, se tiene [JUL78, GAR98b, GAR98c] que para $0 \leq X < M$, siendo M el rango dinámico definido por un conjunto de módulos $\{m_1, m_2, ..., m_N\}$:

$$X = YK + |X|_K \tag{5.1}$$

donde $|X|_K$ es el resto de la división de X entre K, es decir, el residuo de X módulo K, e Y es el cociente de la división X/K. De la ecuación (5.1) se puede deducir el resultado de la operación de escalado, de forma que:

$$Y = \frac{X - |X|_K}{K} \tag{5.2}$$

Por tanto, Y puede calcularse en el sistema de residuos definido por $\{m_1, m_2, ..., m_N\}$; de este modo, el valor de los correspondientes residuos viene dado por:

$$y_i = Y \bmod m_i = \left| \left| X - |X|_K \right|_{m_i} \cdot \left| K^{-1} \right|_{m_i} \right|_{m_i} \tag{5.3}$$

donde $\left|K^{-1}\right|_{m_i}$ es el inverso multiplicativo de K módulo m_i. Así, el escalado queda reducido a una resta y un producto, además de la generación de $|X|_K$. La ecuación (5.3) puede calcularse de manera independiente sobre cada módulo, ya que:

$$\left| X - |X|_K \right|_{m_i} = \left| x_i - \left| |X|_K \right|_{m_i} \right|_{m_i} \tag{5.4}$$

De esta manera, la principal desventaja de este algoritmo es la generación de $|X|_K$; sin embargo, se puede elegir K como el producto de varios de los módulos que definen el RNS sobre el que se trabaja:

$$K = \prod_{i=1}^{S} m_i$$

con $S<N$. De este modo, el inverso multiplicativo de K para m_i ($1 \le i \le S$) no existe, y la ecuación (5.3) deja de tener sentido para $1 \le i \le S$; con ello, y_i ($1 \le i \le S$) no puede calcularse. Sin embargo, es obvio que:

$$0 \le Y < \prod_{i=S+1}^{N} m_i \tag{5.5}$$

La ecuación (5.5) indica que el resultado del escalado queda completamente definido por el conjunto de residuos $[y_{S+1}, \ldots, y_N]$, mientras que $[y_1, \ldots, y_S]$ no pueden ser calculados de la misma manera y deben generarse después del cálculo de $[y_{S+1}, \ldots, y_N]$ a través de extensión de base [JUL78] o cualquier otro algoritmo. Todavía persiste el problema de la generación de $|X|_K$, a menos que $K=\min\{m_i\}$, con lo que no se requiere ningún cálculo adicional para la resolución de (5.4); en concreto, si $m_1=\min\{m_i\} \equiv K$, se puede rescribir la ecuación (5.4) como:

$$\left| X - |X|_K \right|_{m_i} = \left| x_i - \left| x_1 \right|_{m_i} \right|_{m_i} = \left| x_i - x_1 \right|_{m_i} \qquad i = 2, \ldots, N \tag{5.6}$$

y la generación de $|X|_K$ desaparece, dado que es x_1 directamente; de este modo, la ecuación

(5.3) puede calcularse a partir de la representación en residuos de X, excepto y_1.

5.2.2. Implementaciones basadas en tablas de consulta

Debido a (5.5) y a la dificultad para la generación de $|X|_K$, algunos de los algoritmos clásicos [JUL78] para el escalado calculan $[y_{S+1},..., y_N]$ iterativamente. En concreto, Jullien [JUL78] propone dos técnicas diferentes para el escalado, llamadas *algoritmo original* y *técnica de estimación*, respectivamente.

Cuando $S>1$ no tenemos acceso directo a $|X|_K$, de modo que es necesario implementar un proceso iterativo de escalado dado por la recursión:

$$\left|\Phi^{(k+1)}\right|_{m_i} = \left|\left|\Phi^{(k)} - \phi_k\right|_{m_i} \cdot \left|m_k^{-1}\right|_{m_i}\right|_{m_i} \qquad \textbf{(5.7)}$$

donde $\Phi^{(1)}=X$, $\Phi^{(S+1)}=Y$ y $\phi_k=\Phi^{(k)} \bmod m_k$; los límites para la iteración son $1\le k\le S$ y $k\le i\le N$. De este modo, sólo se obtienen con (5.7) los residuos del resultado para $i>S$, $[y_{S+1},..., y_N]$. Para generar el resto de residuos del resultado, $[y_1,..., y_S]$, se puede emplear un proceso de extensión de base empleando el algoritmo MRC [HUA83, JUL78, SZA67]; la representación en bases mixtas, MRC, está definida por:

$$Y = \left(r_0, r_1, r_2, \cdots, r_{D-1}\right) \qquad \textbf{(5.8)}$$

siendo:

$$Y = r_0 + \sum_{i=1}^{D-1} r_i \prod_{j=0}^{i-1} m_j \qquad \textbf{(5.9)}$$

Por tanto, si se define $r_j = \left|R^{(j)}\right|_{m_j+S}$, se puede llevar a cabo la extensión de base a través de la siguiente ecuación recursiva:

$$\left|R^{(k+1)}\right|_{m_i} = \left|\left|R^{(k)} - r_k\right|_{m_i} \cdot \left|m_{k+S}^{-1}\right|_{m_i}\right|_{m_i} \qquad \textbf{(5.10)}$$

donde $R^{(0)}=Y$ y $0\le k\le N-S-1$. A partir de (5.10) se puede calcular los residuos de Y con la

ecuación (5.11).

$$|Y|_{m_i} = \left| \left| -\prod_{k=0}^{N-S-2} m_{k+S} \right|_{m_i} \cdot \left| -R^{(N-S-1)} - \sum_{k=0}^{N-S-2} R^{(k)} \prod_{j=k}^{N-S-2} \left| m_{j+S}^{-1} \right|_{m_i} \right|_{m_i} \right|_{m_i} \tag{5.11}$$

Ésta puede calcularse aplicando repetidamente (5.10).

La implementación directa de las ecuaciones (5.7) y (5.10) con tablas de consulta conduce al llamado *algoritmo original* [JUL78]; estas tablas han de ser de doble entrada, ya que según se desprende de (5.7) y (5.10), la operación fundamental en ambas recursiones es la resta de dos valores y su multiplicación por el inverso multiplicativo de uno de los módulos. De este modo, la operación completa puede precomputarse y almacenarse en estas tablas de doble entrada para cada una de las iteraciones del proceso. Esta implementación puede simplificarse ligeramente introduciendo algunas aproximaciones en las ecuaciones anteriores, lo que produce la llamada *técnica de estimación* que supone una cierta mejora en velocidad y coste respecto del *algoritmo original*. Aunque las ecuaciones (5.7) y (5.10) se ven ligeramente modificadas, la *técnica de estimación* puede implementarse también con tablas de doble entrada. En concreto, ambos algoritmos necesitan L tablas de consulta de dos entradas, siendo:

$$\begin{aligned} L_o &\equiv L_2 + L_3 + L_4 = S\left(N - \frac{S+1}{2}\right) + \frac{(N-S-1)(N-S)}{2} + S(N-S-1) \\ L_e &\equiv L_1 + L_3 + L_4 = (N-S)S + \frac{(N-S-1)(N-S)}{2} + S(N-S-1) \end{aligned} \tag{5.12}$$

donde el subíndice *o* se refiere al *algoritmo original* y *e* a la *técnica de estimación*. Por otra parte, estas tablas han de distribuirse a lo largo de n etapas, con:

$$n_o = N \qquad n_e = N + n_1 - S \quad \left(2^{n_1-1} < S+1 \le 2^{n_1}\right) \tag{5.13}$$

De (5.12) y (5.13) se deduce que la *técnica de estimación* necesita menos tablas que el *algoritmo original* y que éstas se distribuyen en menos etapas, con una latencia total menor. Esto es así ya que:

$$S \geq \frac{S+1}{2} \Rightarrow (N-S)S \leq S\left(N-\frac{S+1}{2}\right) \Rightarrow L_1 \leq L_2 \Rightarrow L_e \leq L_o$$

$$n_1 \leq S \Rightarrow N+n_1-S \leq N \Rightarrow n_e \leq n_o \qquad \textbf{(5.14)}$$

Más concretamente, la *técnica de estimación* necesita n_1 ciclos y $L_1=(N-S)S$ tablas para calcular $[y_{S+1},..., y_N]$, mientras que requiere N-S etapas y L_3+L_4 tablas para la extensión de base.

5.3. NUEVO ALGORITMO BASADO EN MÓDULOS ARITMÉTICOS

Se ha visto en el apartado 5.2.2 una implementación de la ecuación (5.3) basada en tablas de consulta; sin embargo, las ecuaciones (5.3) y (5.6) se reducen a una substracción y un producto, además de la regeneración de y_1 cuando $K=m_1\equiv\min\{m_i\}$; con este valor de K no se requiere ningún tipo de cálculo adicional para generar $|X|_K$. De este modo, se puede diseñar un esquema de escalado consistente en la realización de estas dos operaciones (excluyendo la posterior generación de y_1), dado que la predecible menor velocidad se verá compensada con la mayor simplicidad del circuito y la posibilidad de la construcción de un sistema de propósito general basado en el RNS.

5.3.1. Especificaciones del sistema

Bajo la asunción $K=m_1\equiv\min\{m_i\}$, la ecuación (5.3) puede escribirse como:

$$y_i = Y \bmod m_i = \left| |x_i - x_1|_{m_i} \cdot |K^{-1}|_{m_i} \right|_{m_i} \qquad i = 2, \ldots, N \qquad \textbf{(5.15)}$$

Por tanto, el conjunto de residuos $[y_2,..., y_N]$ puede calcularse a partir de la ecuación (5.15) con una resta y un producto por un coeficiente constante [GAR98c]; de este modo, es posible implementar estas operaciones directamente en lugar de recurrir a una tabla de consulta de doble entrada [JUL78]. La figura 5.1 [GAR98b] muestra el esquema para el

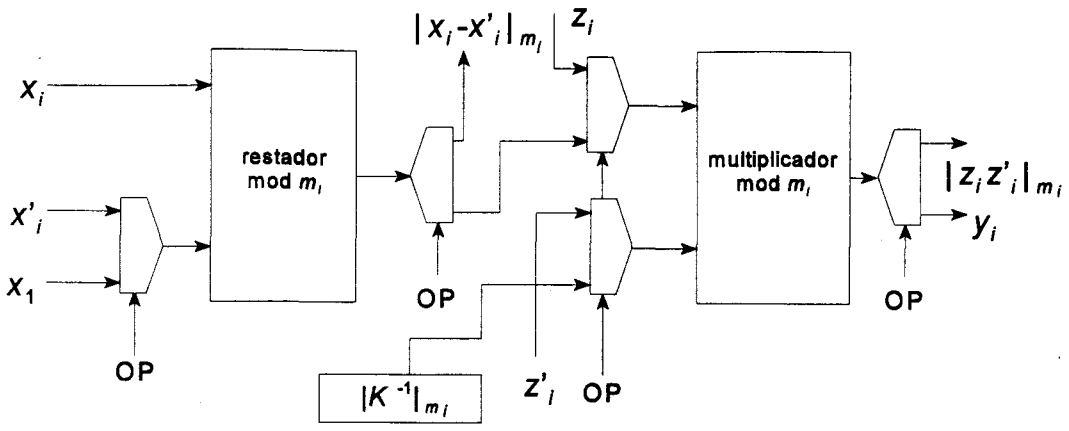


Figura 5.1. Esquema de escalado basado en módulos aritméticos.

módulo m_i para llevar a cabo el escalado en el RNS definido por $\{m_i; i=1,..., N\}$ con $K=m_1=\min\{m_i\}$, según (5.15) y haciendo uso de restadores y multiplicadores convencionales para el RNS. Dado que, como se ha mencionado, la multiplicación es por un coeficiente constante, es posible sustituir el multiplicador en cuestión por una tabla de consulta, lo que todavía supone un considerable ahorro de memoria frente a las alternativas anteriores [GAR98a]; sin embargo, la arquitectura de la figura 5.1 incluye un multiplicador convencional y una serie de multiplexores y demultiplexores a fin de aprovechar al máximo el hardware incluido en este esquema. Así, la señal OP en la figura 5.1 indica si la operación en curso es de escalado o la operación convencional en el restador y el multiplicador, de modo que éste último puede usar $\left|K^{-1}\right|_{m_i}$ (que está almacenado en un registro) como uno de los operandos, o calcular el producto de dos entradas externas. Del mismo modo, los restadores consideran x_1 como uno de los datos de entrada o realizan la operación con dos datos diferentes según el valor de OP. En este caso, la lógica de control adicional para convertir el conjunto de restadores y multiplicadores en un circuito capaz de llevar a cabo el escalado (excepto la generación de y_1) se reduce a unos pocos multiplexores y demultiplexores y registros para almacenar los valores de $\left|K^{-1}\right|_{m_i}$ para $i=2,..., N$; además, en el caso en que la operación no sea el escalado de los datos de entrada, el esquema propuesto puede calcular una resta y un producto de manera independiente.

Es preciso destacar que el esquema propuesto no incluye la regeneración de y_1, que

es un proceso común con todas las alternativas basadas en la forma de escalado expuesta en el apartado 5.2.1; esto es así ya que se pueden usar esquemas basados en tablas de consulta, o bien modificar éstos para obtener arquitecturas similares a la de la figura 5.1, pero en cualquier caso, la comparación que se mostrará en el apartado 5.3.3 no se ve modificada sustancialmente. Más aún, es posible encontrar aplicaciones donde puede suprimirse [GAR98a] este último paso del escalado.

5.3.2. Comparación con otras alternativas

A fin de evaluar las prestaciones del esquema de escalado presentado en el apartado 5.3.1, se va a comparar éste con la *técnica de estimación* [JUL78] introducida por Jullien. Dado que esta última alternativa está completamente basada en el uso de tablas de consulta (v. apartado 5.2.2), se va a comparar nuestra propuesta [GAR98b] con las existentes en términos de la relación entre velocidad y coste definida por:

$$\text{SCR} = \frac{1}{\text{ciclos acceso ROM} \cdot \text{bits ROM}} \qquad \textbf{(5.16)}$$

Para emplear la ecuación (5.16) es necesario asumir que todos los elementos aritméticos presentes en esta nueva alternativa están construidos con ROM; sin embargo, esta asunción [GAR98b] puede realizarse sin demasiados problemas, ya que para esta comparación se supondrá que el restador está construido según el método de rotación-selección (v. apartado 3.1.2), y que el multiplicador sigue el diseño [GAR97a] propuesto en el capítulo IV. De este modo, es evidente que el tiempo de cálculo para el multiplicador es de tres ciclos de acceso a memoria, mientras que la cantidad de memoria exigida para su realización es fácilmente calculable. Para estimar la latencia en términos de ciclos de acceso a memoria y la cantidad de memoria requerida por el restador módulo m de rotación-selección, según la estructura de la figura 3.3, con $k = \lceil \log_2 m \rceil$, éste requiere [LAK94, GAR97b] dos decodificadores de k a m, una ROM de m palabras de k bits, y la lógica de rotación de $m \times m$ celdas. Se considerarán los decodificadores de k a m y la lógica de rotación como una memoria ROM, con una matriz $m \times m$ de celdas con direccionamiento bidimensional; por tanto, el coste del restador en términos de bits de

memoria puede evaluarse como:

$$\text{bits ROM} = m^2 + m\,k \qquad \textbf{(5.17)}$$

Al mismo tiempo, se puede considerar que el tiempo de cálculo para este circuito es de dos ciclos de acceso a memoria, uno para la decodificación de las entradas y la rotación y otro más para la codificación del resultado de la rotación. Es obvio que esto es solamente una cota superior y que tanto coste como tiempo de cálculo serán menores; sin embargo, se hará uso de estos valores ya que serán útiles para la comparación con las propuestas anteriores.

5.3.3. Resultados experimentales

La tabla 5.1 recoge los resultados en términos de bits de memoria y SCR (5.17) para la *técnica de estimación* [JUL78] y el esquema de la figura 5.1 [GAR98b], al tiempo que la tabla 5.2 muestra los diferentes submódulos (ha de recordarse que el multiplicador de la figura 5.1 se implementa según la técnica de mapeo isomórfico submodular descrita en el capítulo IV) y cantidades de memoria requeridas para cada uno de los módulos que intervienen en la tabla 5.1.

TABLA 5.1

$\{m_i\}$	Técnica de estimación		Módulos aritméticos convencionales		SCR_2/SCR_1	SCR_2/SCR_1 (pipeline)
	bits	SCR_1 (Kbit^{-1})	*bits*	SCR_2 (Kbit^{-1})		
23, 29, 31	10240	0.10000	5890	0.03477	0.348	1.739
43, 53, 61	49152	0.02068	14829	0.01381	0.668	3.339
109, 113, 127	229376	0.00446	60026	0.00341	0.764	3.821
233, 239, 241	1048576	0.00098	208863	0.00098	1.004	5.020
19, 23, 29, 31	15360	0.06667	7854	0.02608	0.391	1.956
41, 43, 53, 61	73728	0.01389	18292	0.01120	0.806	4.031
103, 109, 113, 127	344064	0.00298	80352	0.00255	0.856	4.282
211, 233, 239, 241	1572864	0.00065	263264	0.00078	1.195	5.974

TABLA 5.2

Módulo	Submódulos	Bits multiplicador	Bits restador
19	2, 9	1508	456
23	2, 11	1508	644
29	4, 7	704	986
31	2, 3, 5	932	1116
41	5, 8	1536	1927
43	2, 3, 7	1380	2107
53	4, 13	2208	3127
61	3, 4, 5	1920	4087
103	2, 3, 17	8996	11330
109	4, 27	7840	12644
113	7, 16	3904	13560
127	2, 7, 9	5060	17018
211	14, 15	8192	46209
233	8, 29	11456	56153
239	2, 7, 17	14020	59033
241	15, 16	8192	60009

La tabla 5.1 se ha calculado asumiendo que la técnica de estimación emplea un ciclo de acceso a memoria para el cómputo de $[y_2,..., y_N]$, mientras que el esquema de la figura 5.1 requiere cinco ciclos de acceso a memoria. De la tabla 5.1 se deduce que el esquema propuesto en la figura 5.1 para el escalado en el RNS conduce a una mejor relación entre velocidad y coste (SCR) para módulos grandes que los algoritmos existentes. Más concretamente, el esquema de la figura 5.1 implica unas mejores prestaciones ($SCR_2/SCR_1>1$ en la tabla) para conjuntos de módulos representados por 8 o más bits; al mismo tiempo, este nuevo esquema conduce a resultados muy próximos a los de los algoritmos existentes para módulos menores.

Por otra parte, la tabla 5.1 ha sido calculada asumiendo cinco ciclos de acceso a memoria para realizar el escalado con este esquema basado en módulos aritméticos. Pero se puede dotar de segmentación de cauce a esta alternativa aquí propuesta, como se deduce fácilmente de la figura 5.1, usando multiplicadores con segmentación de cauce (v. capítulo IV) e implementando la segmentación en los restadores de la sección 3.1.2. De este modo, se obtiene un nuevo resultado en cada ciclo de reloj, que es la misma velocidad que con los algoritmos anteriores, que a su vez pueden ser también dotados de segmentación de

cauce; así se mejora en un factor cinco la relación SCR, lo que conduce a mejores prestaciones que la *técnica de estimación* para todos los conjuntos de módulos propuestos.

Finalmente, la comparación de la tabla 5.1 no recoge el hecho de que el esquema de la figura 5.1 conduce a un sistema que lleva a cabo tres operaciones diferentes en el RNS con el mismo hardware. Además, como se vio en la sección 3.1.2, el restador empleado en esta propuesta puede ser transformado fácilmente en un sumador/restador, con lo que el esquema de la figura 5.1 puede realizar las cuatro operaciones básicas en el RNS -suma, resta, multiplicación y escalado (excluyendo la regeneración de y_1)- en función de dos señales de control: OP (figura 5.1), que indica que ha de realizarse el escalado, y otra señal que indica el tipo de operación a llevar a cabo en el sumador/restador cuando no se está calculando el escalado.

5.4. NUEVO ALGORITMO BASADO EN TABLAS DE CONSULTA

Con los algoritmos descritos en la sección 5.3, la memoria requerida y el tiempo de cálculo aumentan con el número de módulos empleados en el RNS. En esta sección se demostrará que, bajo ciertas condiciones, es posible reducir los requerimientos de memoria al mismo tiempo que se obtiene un sistema más rápido.

5.4.1. Algoritmo de escalado en dos ciclos de acceso a memoria

Cuando K es producto de varios de los módulos, se ha visto en el apartado 5.2.1 cómo el escalado puede descomponerse en el cálculo de $[y_{S+1},..., y_N]$ y la generación de $[y_1,..., y_S]$. Más aún, de la ecuación (5.3) se desprende [GAR97c] que y_i $(S+1 \le i \le N)$ queda determinado de manera unívoca por x_i y por $[x_1,..., x_S]$. De este modo, y_i $(S+1 \le i \le N)$ puede calcularse a través de tablas de consulta a partir de $[x_1,..., x_S, x_i]$; es decir, se puede almacenar el valor de y_i $(S+1 \le i \le N)$ en una memoria direccionada por $[x_1,..., x_S, x_i]$. Por otra parte, tal y como se ha comentado anteriormente, $[y_1,..., y_S]$ quedan determinados de manera unívoca por $[y_{S+1},..., y_N]$. Por tanto, se puede llevar a cabo la operación de escalado

tal y como se muestra en la figura 5.2, en sólo dos ciclos de acceso a memoria y con *N-S* tablas de *S*+1 entradas y una tabla de *N-S* entradas (en realidad, esta última tabla corresponde a *S* tablas de *N-S* entradas). El algoritmo puede concretarse en el siguiente pseudocódigo:

```
begin
     for i:=(S+1) to N do begin
          consulta en tabla de y_i con [x_1,...,x_S,x_i]
     end
     for i:=1 to S do begin
          consulta en tabla de y_i con [y_S+1,...,y_N]
     end
end
```

Con lo anterior, la operación de escalado [GAR97c] queda dividida en dos operaciones independientes; además, ambas pueden llevarse a cabo de manera paralela para cada *i*, es decir, para cada módulo; de este modo, el escalado en su conjunto sólo requiere dos ciclos de acceso a memoria. Este esquema es válido sólo para pequeños valores de *N* y *S*, a menos que sea aceptable un elevado uso de memoria, dado que tablas con más de dos entradas pueden resultar excesivamente costosas. En cualquier caso, existen ciertos conjuntos de módulos para los que el esquema de la figura 5.2 es viable. El primero de ellos es cualquier conjunto de tres módulos, con uno de ellos como constante de escalado *K*; por ejemplo, el conjunto {29, 31, 32} define un rango de 14 bits, que puede ser suficiente para ciertas aplicaciones [GAR97c]. Si se define *K* como uno de los módulos, el escalado descrito por la figura 5.2 sólo necesita el uso de 3 tablas de doble entrada, dos de ellas para el cálculo de y_2 e y_3, y otra más para la generación de y_1 a partir de y_2 e y_3; por otra parte, los algoritmos descritos en la sección 5.2.2 [JUL78] requieren n=3 ciclos y L=4 tablas. Se podrá aplicar este tipo de conjunto de módulos en sistemas que operen con datos en el rango definido por uno de los módulos; así, un producto y una operación de escalado producirán un resultado en el rango original, pero realizando estas operaciones sobre cada módulo de manera independiente. Además, para módulos mayores y sistemas que necesiten un mayor valor de *K*, pueden emplearse dos operaciones

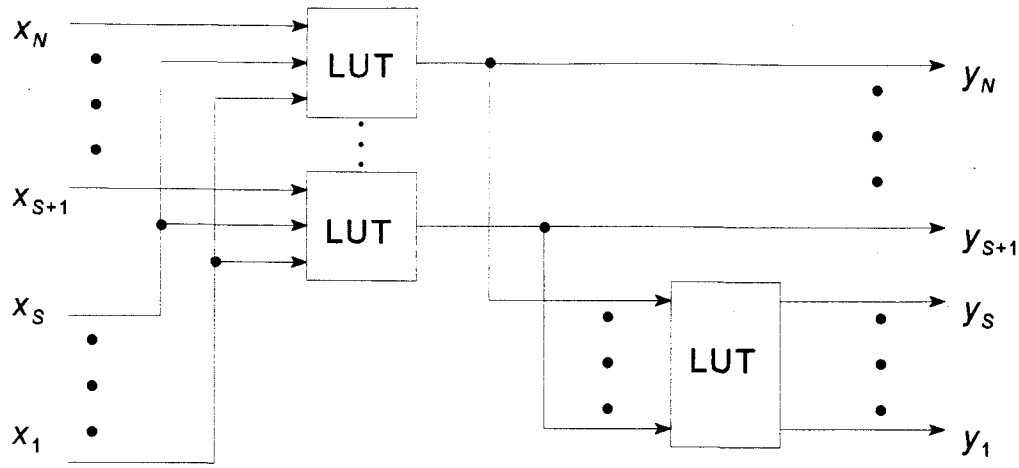


Figura 5.2. Escalado en el RNS en dos ciclos de acceso a memoria.

consecutivas de escalado con uno solo de los módulos como constante de escalado; de este modo, aunque el tiempo de cálculo será el equivalente a cuatro ciclos de acceso a memoria, el crecimiento exponencial de las memorias con el tamaño de los módulos conducirá a una alternativa más ventajosa que las proporcionadas por propuestas anteriores y K implicando más de un módulo. Aunque el escalado con $K<M^{1/2}$ puede parecer poco realista desde un punto de vista general, la relación entre K y M estará solamente fijada por los datos procesados y el tipo de procesamiento a que éstos sean sometidos, tal y como se ha discutido anteriormente.

Existen situaciones en las que tablas de más de dos entradas pueden resultar inadecuadas por su elevado coste, a pesar de que el tiempo de cálculo sea siempre de dos ciclos de acceso a memoria. Por ejemplo, en el RNS formado por {27, 29, 31, 32} y con K=27×29 (N=4 y S=2), puede demostrarse que n=3 (5.13) y L=8 (5.12) con la *técnica de estimación*, mientras que el esquema de la figura 5.2 necesita dos tablas de tres entradas (32K×5 bits) para el cálculo de y_3 e y_4, y dos tablas de doble entrada (1K×5 bits) para la generación de y_1 e y_2. De hecho, esta cantidad de memoria puede emplearse sin demasiados problemas, pero puede resultar excesiva para algunas aplicaciones.

La arquitectura propuesta puede emplearse [GAR97c] en el desarrollo de sistemas basados en el RNS con conjuntos de módulos pequeños; de este modo, se obtiene un

escalado rápido de una manera muy simple, que puede ser muy adecuado para su uso en implementaciones VLSI o con FPGAs. Aunque el tamaño creciente de las memorias requeridas puede ser un factor limitante para módulos grandes, o conjuntos de módulos con un elevado número de éstos, este esquema de escalado supone una alternativa rápida, ya que el tiempo de cálculo queda reducido a dos ciclos de acceso a memoria.

5.4.2. Algoritmos mixtos para el escalado basado en tablas de consulta

Como se ha comentado anteriormente, la cantidad de memoria requerida puede limitar la utilidad del algoritmo anterior para la aplicación del escalado en dos ciclos de acceso a memoria; esto es así debido a la posible presencia de tablas de más de dos entradas, a pesar de que la latencia del algoritmo en términos de ciclos de acceso a memoria se mantiene fija. A fin de encontrar un equilibrio [GAR97c] entre velocidad de cálculo y requerimientos de memoria en aplicaciones donde las tablas deben poseer un tamaño limitado, se pueden combinar de dos maneras sencillas los algoritmos iterativos (v. sección 5.2.2) con la nueva arquitectura propuesta en el apartado 5.4.1:

1. Cálculo de $[y_{S+1},..., y_N]$ con la *técnica de estimación* [JUL78], lo que implica el uso de L_1 tablas de consulta y n ciclos de acceso; y generación de $[y_{...1}, y]_S$ con S tablas de N-S entradas. Se designará esta opción como *generación por consulta de* $[y_1,..., y_S]$.

2. Cálculo de $[y_{S+1},..., y_N]$ con N-S tablas de S+1 entradas, y generación de $[y_1,..., y_S]$ con el algoritmo MRC de extensión de base [JUL78], lo que requiere el uso de L_3+L_4 tablas de dos entradas (5.12) y N-S ciclos. Se llamará a esta opción *cálculo por consulta de* $[y_{S+1},..., y_N]$.

Estas dos opciones se muestran en la figura 5.3. De este modo, es posible reducir la cantidad total de memoria requerida por el algoritmo de consulta pura del apartado 5.4.1 al sustituir las etapas más gravosas por alguna de las estructuras iterativas introducidas por otros autores [JUL78]. Esto es así debido a que la parte del algoritmo que implica tablas

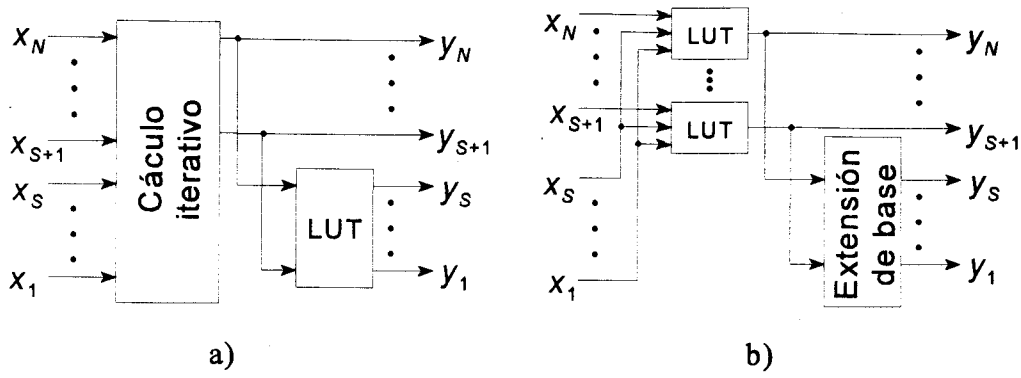


Figura 5.3. Uso de algoritmos iterativos con métodos de consulta. a) Generación por consulta de $[y_1,..., y_S]$. b) Cálculo por consulta de $[y_{S+1},..., y_N]$.

de consulta con un excesivo número de entradas es sustituida por la correspondiente estructura iterativa; ésta implementa el cálculo de la ecuación (5.7) para el algoritmo de *generación por consulta de* $[y_1,..., y_S]$, mientras que corresponde a la recursión definida por (5.10) para el *cálculo por consulta de* $[y_{S+1},..., y_N]$. Sin embargo, la latencia del esquema de escalado resultante se ve aumentada en la medida que una sola etapa de acceso a memoria es sustituida por la correspondiente estructura iterativa.

5.4.3. Comparación con otras alternativas

La tabla 5.3 muestra la comparación para el número de tablas, L, número de ciclos de acceso a memoria, n, y número de bits de memoria para varios conjuntos de módulos y valores de K para las diferentes alternativas presentadas en esta sección. Para cada uno de los conjuntos de módulos y K propuestos se muestran los resultados para la *técnica de estimación* (v. apartado 5.2.2), el algoritmo de consulta en dos ciclos de acceso a memoria propuesto en el apartado 5.4.1, y los algoritmos mixtos presentados en el apartado anterior. De esta manera, es posible mantener un uso reducido de memoria con una velocidad razonable de cálculo; por ejemplo, para el conjunto {27, 29, 31, 32} y K=27×29, la *generación por consulta de* $[y_1,..., y_S]$ implica un ahorro considerable de memoria con respecto al algoritmo de consulta pura de dos ciclos de acceso a memoria; pero al mismo tiempo es una solución más rápida y económica que la *técnica de estimación*. Para cada uno de los conjuntos de módulos propuestos, la tabla 5.3 muestra cómo una de las técnicas

mixtas, *cálculo por consulta de* $[y_{S+1},..., y_N]$ o *generación por consulta de* $[y_1,..., y_S]$, es siempre una solución intermedia entre los algoritmos de la sección 5.2.2 y el esquema de consulta pura de la figura 5.2; además, uno de ellos es una solución mejor que la *técnica de estimación*, en términos de memoria empleada y velocidad, para la mayoría de combinaciones de módulos y *K*.

TABLA 5.3

$\{m_i\}$	K	**Técnica de estimación (5.2.2)**	**Consulta pura (5.4.1)**	**Generación por consulta de** $[y_1,..., y_S]$	**Cálculo por consulta de** $[y_{S+1},..., y_N]$
		***n, L* memoria**	***L* (*n*=2) memoria**	***n, L* memoria**	***n, L* memoria**
29, 31, 32	29	3, 4 20K	3 15K	2, 3 15K	3, 4 20K
29, 31, 32	29×32	3, 3 15K	3 160K	3, 4 10K	2, 1 160K
27, 29, 30, 31	27	4, 8 40K	4 175K	2, 4 175K	4, 8 40K
27, 29, 30, 31	27×29	4, 8 40K	4 330K	3, 6 30K	3, 5 335K
25, 27, 29, 31, 32	29	5, 13 65K	5 5140K	2, 5 5140K	5, 13 65K
25, 27, 29, 31, 32	29×31	5, 14 70K	5 800K	3, 8 350K	4, 10 515K
25, 27, 29, 31, 32	25×29×31	4, 11 55K	5 10255K	3, 9 45K	3, 6 10260K

5.5. EJEMPLO DE DISEÑO VLSI

Para ilustrar el algoritmo de escalado en dos ciclos de acceso a memoria propuesto en el apartado 5.4.1, a continuación se van a exponer los detalles de una implementación VLSI del mismo para un conjunto de tres módulos de 5 bits. En los siguientes apartados se irán desglosando los detalles de diseño de la ROM de 1024×5 bits, que es el bloque constitutivo de este esquema de escalado. Esta implementación ha sido realizada con el

proceso CMOS14TB (v. apéndice C), un proceso CMOS de 0.6μm (longitud efectiva del canal de 0.5μm), tres capas de metal y una de polisilicio.

5.5.1. Descripción general del sistema

Tal y como se ha descrito en el apartado 5.4.1, el esquema de escalado propuesto está completamente basado en el uso de tablas de consulta; de este modo, para un RNS definido por tres módulos con uno de ellos como constante de escalado, ésta operación puede realizarse con tres tablas de doble entrada. Por tanto, el problema del escalado se reduce al diseño de una memoria ROM eficiente. Así, para tres módulos de 5 bits, son necesarias tres ROMs de 1024×5 bits, con lo que la creación de este circuito será el objetivo principal de esta sección.

Entre las diversas alternativas [UYE92] existentes para la construcción de una memoria ROM, se han seguido las directrices desarrolladas por el VLSI Research Group de la Universidad de Windsor, a fin de conseguir un memoria de alta velocidad y reducido consumo de energía. De este modo, las celdas constitutivas de cada memoria son:

- `cell_2x8`: la celda básica de memoria es un transistor NMOS que conecta o no la línea de bit con GND en función de que exista o no un contacto con el que se programa la celda en cuestión. De este modo, la matriz de celdas de memoria se consigue replicando la celda básica, que contiene dos filas de ocho celdas.

- `dec256`: la matriz de celdas de memoria consta de 64 filas (32 celdas `cell_2x8`), de forma que 32 celdas `dec256` producen las correspondientes señales de selección de fila a partir de seis de los diez bits de dirección de la memoria.

- `sel16_1`: dado que las señales de selección de fila afectan a toda la matriz de celdas, de cada grupo de 16 bits de la fila seleccionada se enruta uno hacia la línea de salida a través de la celda `sel16_1`, que hace uso de los cuatro bits restantes

de la dirección. De este modo, una matriz 32×2 de celdas `cell_2x8` puede almacenar el contenido de cada bit de la palabra de salida, y cinco de estas matrices con líneas de selección de fila y columna comunes almacenan la palabra completa.

- `amp_inv`: una vez que se ha seleccionado una fila y una de las 16 líneas de bit, la salida queda conectada con el transistor correspondiente de la celda `cell_2x8` apropiada. Esta celda precarga un nodo interno a Vdd, de forma que si este transistor no conectaba la línea de datos con GND, este 1 lógico se invierte y se da como salida un 0 lógico; por contra, si el transistor en cuestión conecta la línea de datos con GND, la precarga queda anulada y el inversor proporciona un 1 lógico como salida. En cualquier caso, el valor generado se almacena a través de una puerta de paso CMOS.

- `add_buf`: finalmente, esta celda almacena la dirección de la memoria a través de puertas de paso y genera los complementos lógicos de los mismo y de la señal de reloj de la memoria, para su distribución por la misma.

Entre las razones que llevaron a la elección de esta estructura para la memoria están:

- Dado que la salida se genera por precarga de un nodo interno o por descarga del mismo por conexión a GND, tanto `cell_2x8` como `sel16_1` están formadas sólo por transistores NMOS, ya que sólo es necesario transmitir ceros lógicos; por tanto, es posible realizar una memoria muy compacta al no tener que coexistir pozos p y n en la mayor parte de la misma.

- Por otra parte, la programación de la memoria es muy sencilla y puede hacerse a través de ficheros CIF (Caltech Intermediate Form), de muy fácil generación, sin más que añadir un contacto en los transistores que deban almacenar un 1 lógico y dejar el transistor sin conexión para producir un 0.

- Del mismo modo, la celda `dec256` se usa para la selección de las 64 filas sin más que usar 32 de estas celdas y programarlas de manera adecuada con un fichero CIF;

- Dado que la memoria incorpora puertas de paso para las direcciones y la salida, no es necesario el empleo de recursos adicionales para su inclusión en estructuras segmentadas.

A continuación, se va a describir en detalle cada una de estas celdas.

UNIVERSIDAD DE GRANADA
13 MAYO 1998
COMISION DE DOCTORADO

5.5.2. Celdas de memoria

La estructura de la celda `cell_2x8` queda reflejada en la figura 5.4, asumiendo que todos los transistores son de igual tamaño, con L=0.6μm y W=1.6μm. De este modo, las celdas B1_7 y B0_7 corresponden a la misma línea de datos, de forma que sólo una de ellas estará seleccionada en función de las señales de selección de fila (ROW0 y ROW1); por otra parte, la existencia de un contacto entre el drenador del transistor correspondiente y la línea de datos conectará ésta a GND, produciendo un 1 en la salida de la memoria; por el contrario, si no existe este contacto para el transistor seleccionado, el nodo interno anterior a la salida se precarga a 1 y la salida es un 0 lógico.

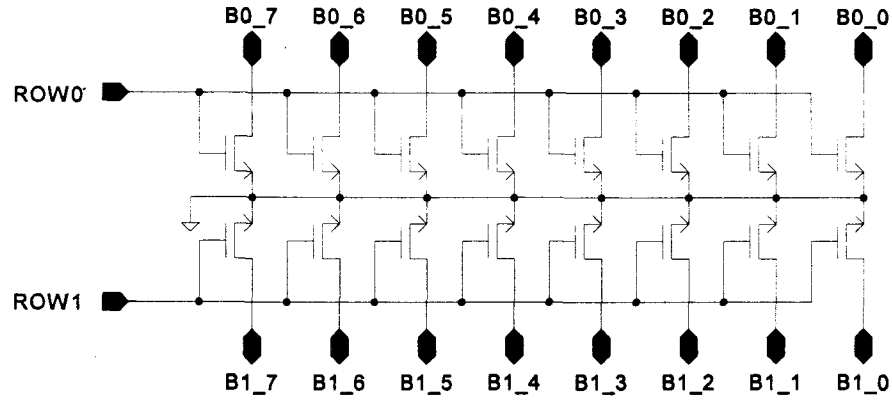


Figura 5.4. Esquemático de la celda `cell_2x8`, con 16 transistores NMOS de 0.6μm×1.6μm.

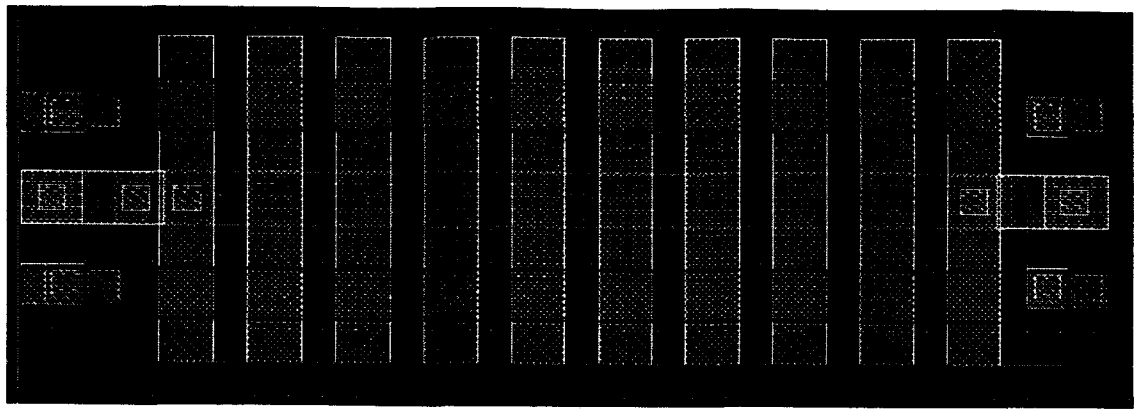


Figura 5.5. Layout de la celda `cell_2x8`.

La figura 5.5 muestra el layout de la celda `cell_2x8`, que es muy compacta. Para la definición de los transistores sólo es necesaria una isla n horizontal, que es la fuente común de todos los transistores y está conectada a GND a través de las líneas verticales correspondientes en los extremos de la celda, y 8 islas n verticales que cruzan a la horizontal. Las puertas de los transistores están determinadas por sendas líneas de polisilicio que distribuyen las señales de selección de fila. A fin de reducir el retardo en la distribución de estas señales, existen dos líneas adicionales de metal_2, con conexiones periódicas entre éstas y las correspondientes líneas de polisilicio. Finalmente, ocho líneas verticales de metal_1 cubren los correspondientes transistores y corresponden a las líneas de datos de la memoria; así, un simple contacto metal_1-poli permite programar un 1 en dicha posición de memoria. Por otra parte, las dos líneas de GND en los extremos de la celda distribuyen esta señal por toda la matriz de celdas para polarizar las islas que forman la fuente común en cada celda `cell_2x8`, así como para polarizar el sustrato periódicamente con islas p.

5.5.3. Selección de filas

Tal y como se ha comentado en el apartado anterior, cada copia de `cell_2x8` contiene dos filas de celdas de memoria; de este modo, es lógico pensar que cada copia de `dec256` genere las dos señales para una fila completa de celdas `cell_2x8`. La figura 5.6 recoge el esquema básico para la celda encargada de sintetizar la señal de selección para dos filas, de forma que 32 de estas celdas pueden discernir entre las 64 direccionadas

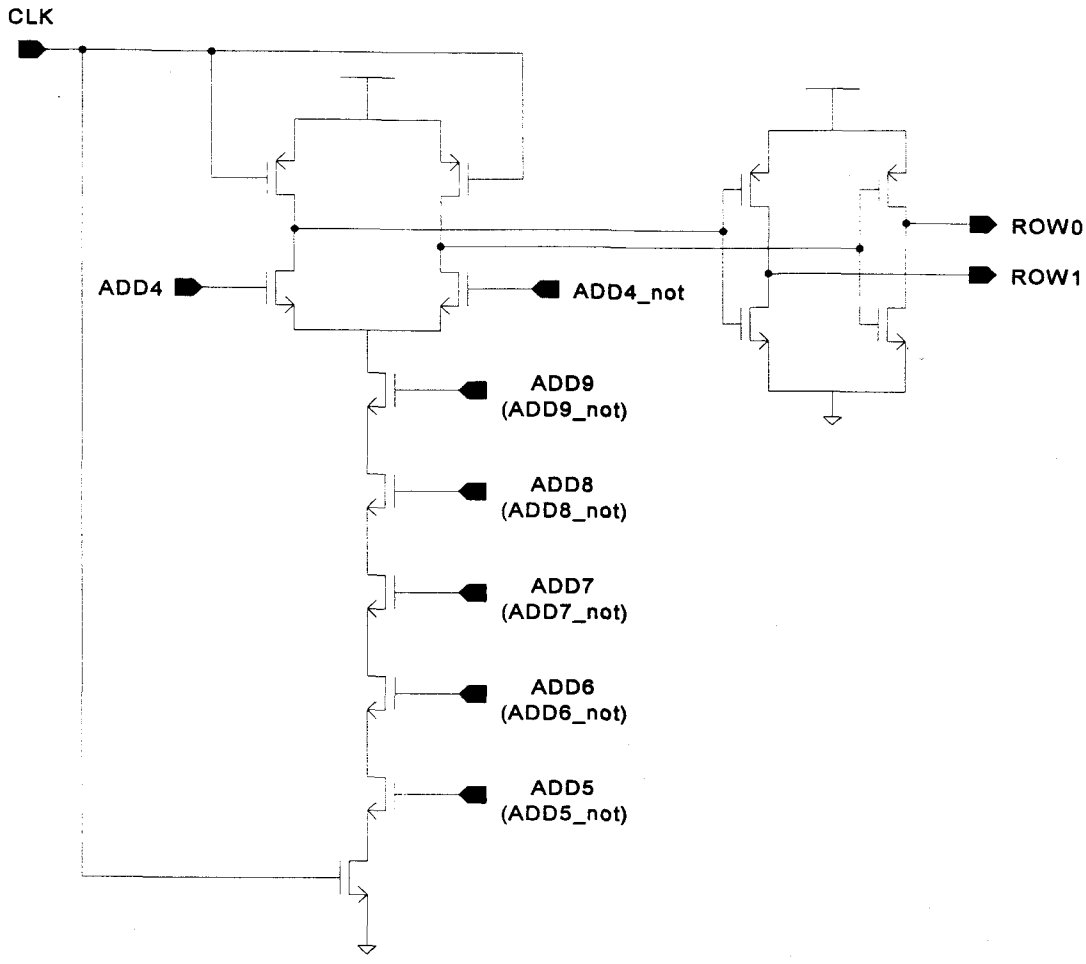


Figura 5.6. Esquemático de la celda `dec256`.

por los seis bits más significativos de dirección de la memoria. El funcionamiento del circuito de la figura 5.6 es sencillo, y consta de 8 transistores NMOS (0.6μm×1.6μm) y dos transistores PMOS (0.6μm×2.0μm) para la precarga de los nodos internos, y dos inversores que generan las correspondientes señales de selección de fila, con transistores NMOS de área 0.6μm×1.6μm y PMOS de 0.6μm×2.6μm. De este modo, cuando la combinación de entradas hace que una de las columnas de transistores NMOS entre en conducción durante el ciclo activo del reloj, los nodos internos precargados a 1 durante el ciclo inactivo de CLK quedan conectados a GND, de forma que el correspondiente inversor inyecta un 1 en la línea de selección de fila; en caso contrario, los nodos no son descargados y los inversores mantienen las líneas ROW0 y ROW1 a cero.

Por otra parte, a fin de reducir el área total, sólo los transistores para el bit menos significativo están duplicados, mientras que para el resto de entradas existe un sólo

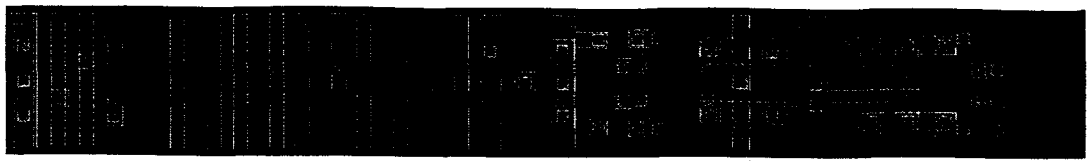


Figura 5.7. Layout de la celda `dec256`.

transistor que está conectado a la línea correspondiente según la posición de la celda `dec256`. Además, así se evitan descargas erróneas por repartición de carga, evitando la situación en que todos los transistores de una de las columnas condujeran, excepto el inferior; en este caso, aunque el nodo interno no debe descargarse, se produciría un reparto de carga entre los transistores activos, que podrían hacer que la línea de selección de fila correspondiente fuera activada.

La figura 5.7 recoge el layout de `dec256`, donde se observa que sólo las líneas de dirección del bit menos significativo del grupo, a la izquierda de la figura, están conectadas y existen dos transistores diferenciados; sin embargo, sólo es necesario definir un fichero CIF con los contactos adecuados para programar la columna de 32 celdas `dec256`. Se observa que las líneas de dirección y sus complementos, GND, Vdd y CLK cruzan verticalmente el layout (metal_1), mientras que líneas horizontales de metal_2 facilitan la comunicación entre los nodos precargados y los inversores correspondientes.

5.5.4. Selección de columnas

Tal y como se mencionó anteriormente, se emplean seis bits de dirección para la selección de filas, mientras que los cuatro restantes se dedican para la selección de columnas; de este modo, se debe de elegir uno de cada 16 bits. Por otra parte, tal y como se viene comentando, las celdas que almacenen un 1 conectan la línea correspondiente con GND, descargando un nodo interno precargado y generando un inversor el 1 correspondiente; por el contrario, si la celda no tiene contacto (v. apartado 5.5.2), almacena un 0, y realmente no altera el nodo interno. Por tanto, sólo es necesario transmitir la conexión a GND para la selección de columnas, con lo que es posible emplear un sencilla red de transistores NMOS, como muestra la figura 5.8, con el

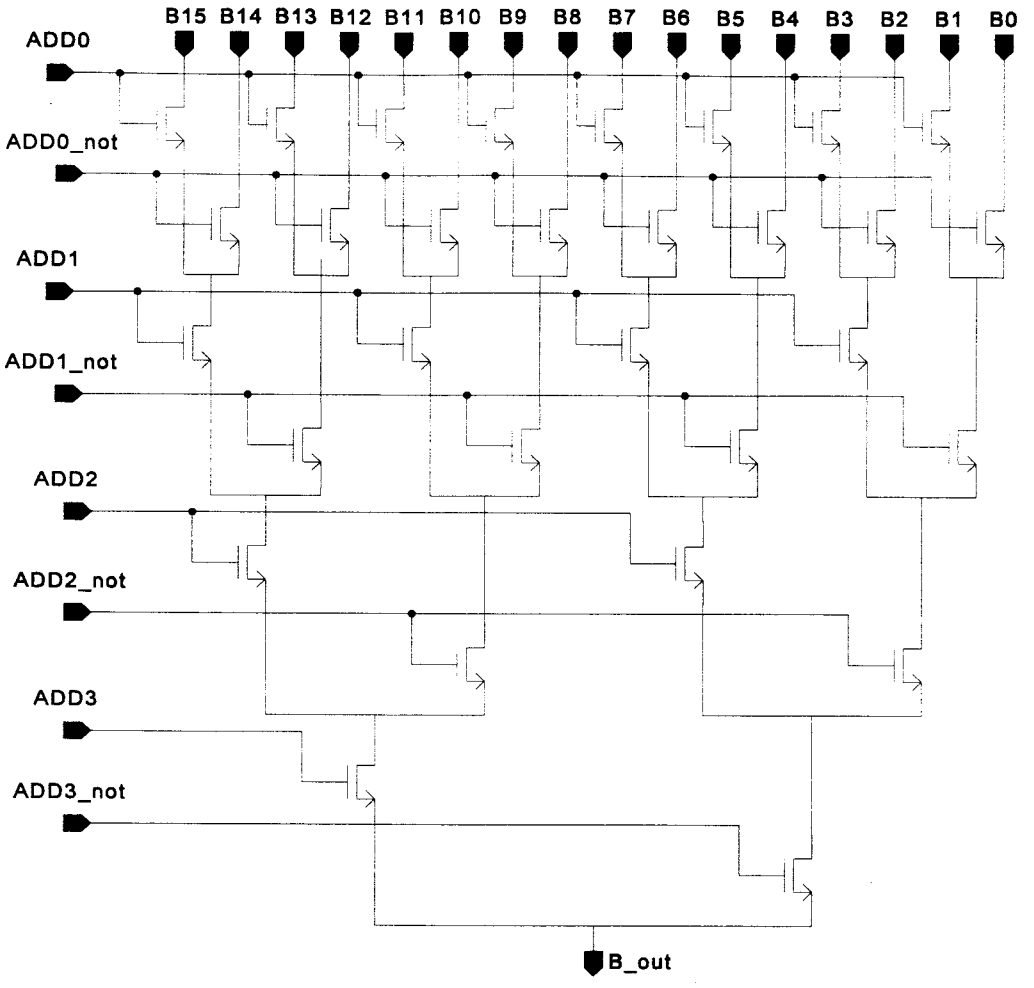


Figura 5.8. Esquemático de la celda `sel16_1`.

esquemático de la celda `sel16_1`. Así, sólo es necesario aplicar los bits de dirección correspondientes para conectar la posible celda con conexión a GND con el nodo interno anterior al inversor de salida.

La figura 5.9 muestra el layout de `sel16_1`, con transistores NMOS de área 0.6μm×3.6μm. Dado que son únicamente necesarios transistores NMOS, el diseño es muy compacto y encaja con dos celdas `cell_2x8` (figura 5.5), con lo que las líneas de GND de esta última celda ayudan a distribuir esta señal para la polarización del sustrato. Así, las conexiones entre los diferentes transistores NMOS y las líneas que distribuyen GND están realizadas con metal_1, mientras que líneas horizontales de polisilicio definen las puertas de los diferentes transistores y están conectadas a las correspondientes señales de dirección. Sin embargo, al igual que para `cell_2x8` y a fin de minimizar el retardo y la disipación en la propagación de estas señales, estas líneas de polisilicio están trazadas

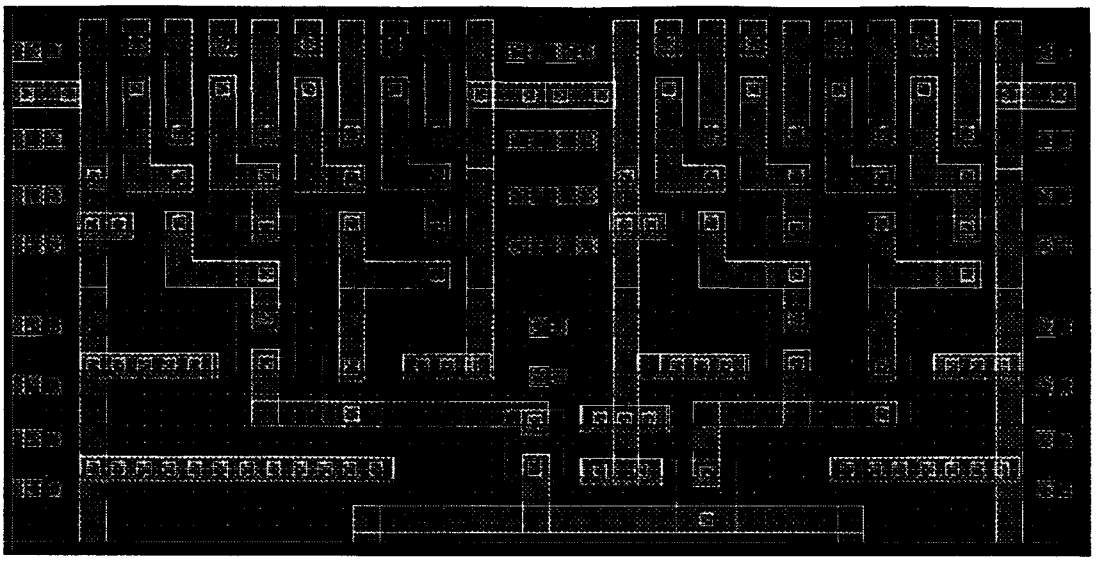


Figura 5.9. Layout de la celda `sel16_1`.

junto a líneas de metal_2 con conexiones periódicas.

Por otra parte, una vez que se ha realizado la selección de filas y se ha seleccionado el bit correspondiente con `sel16_1`, es necesario incluir una nueva celda que realice la precarga del nodo anterior a la salida y lo invierta. Esto se consigue con el circuito de la figura 5.10, `amp_inv`, que además incluye una puerta de paso en la salida; en esta celda todos los transistores NMOS son de tamaño 0.6μm×1.6μm, mientras que los PMOS ocupan 0.6μm×2.6μm. De este modo, si el nodo B_out de las figuras 5.8 y 5.10 se halla conectado a GND a través del transistor seleccionado por los bits de dirección en la celda `cell_2x8` correspondiente, el inversor de la figura 5.10 genera un 1 en la salida. Por el

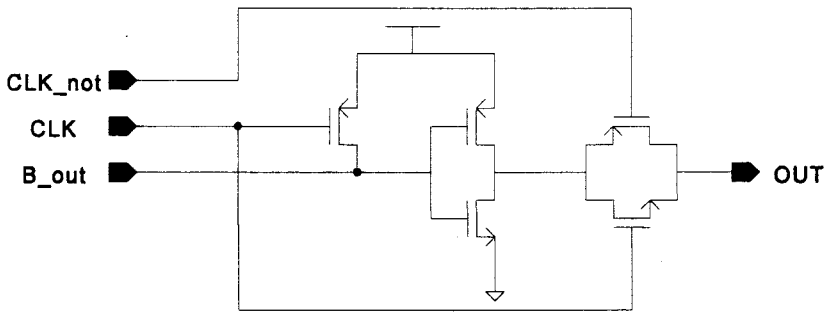


Figura 5.10. Esquemático de la celda `amp_inv`.

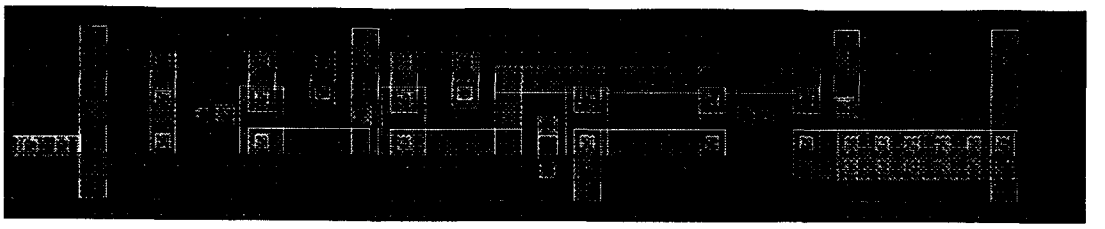


Figura 5.11. Layout de la celda `amp_inv`.

contrario, si el transistor seleccionado no posee el contacto en la máscara de programación, B_out queda flotante durante el ciclo activo de CLK y la precarga anterior se mantiene, generándose un 0 en la salida. Por último, la puerta de paso mantiene este valor durante el ciclo negativo de CLK. Se puede observar el layout de esta celda en la figura 5.11, donde las líneas horizontales de metal_2 distribuyen las señales CLK, CLK_not y Vdd por toda la celda, mientras que se aprovechan las líneas de GND provenientes de la celda `sel16_1` para el inversor y la polarización del sustrato.

5.5.5. Buffer de direcciones

En los apartados anteriores se ha descrito la estructura básica de la memoria.

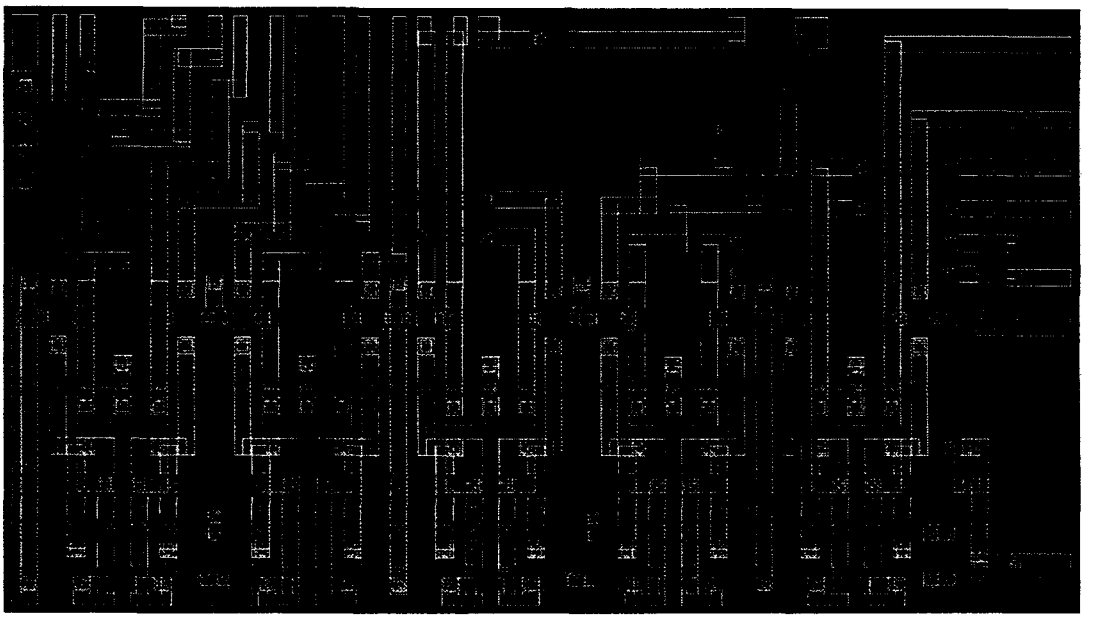


Figura 5.12. Layout de la celda `add_buf`.

Resulta necesario generar los complementos de los diferentes bits de selección y de la señal CLK, así como distribuir GND y Vdd por las diferentes celdas. De todo esto se encarga la celda `add_buf`, cuyo layout se muestra en la figura 5.12. Básicamente, esta celda incorpora una puerta de paso y un inversor para cada bit de dirección, enrutando éstos y sus complementos hacia las celdas correspondientes (selección de filas o columnas); además, genera el complemento de CLK, CLK_not, y distribuye estas dos señales, GND y Vdd por las diferentes celdas. La celda `add_buf` está diseñada para encajar debajo de la columna de 32 celdas `dec256` y a la izquierda de la fila de celdas `sel16_1` y `amp_inv`.

La construcción de una memoria de 1024×5 bits requiere 32 celdas `dec256` dispuestas en una columna, una matriz 32×10 de celdas `cell_2x8` situada a la derecha de la primera y cinco parejas de celdas `sel16_1` y `amp_inv` bajo la matriz de celdas de memoria a fin de generar los cinco bits de la palabra de salida. Finalmente, en la esquina inferior izquierda se ubica la celda `add_buf`. La figura 5.13 recoge un detalle de la esquina inferior izquierda de una ROM de 1024×5 bits mostrando todas las celdas mencionadas. El tamaño total de la ROM de 1024×5 bits es 399.2μm×355.7μm.

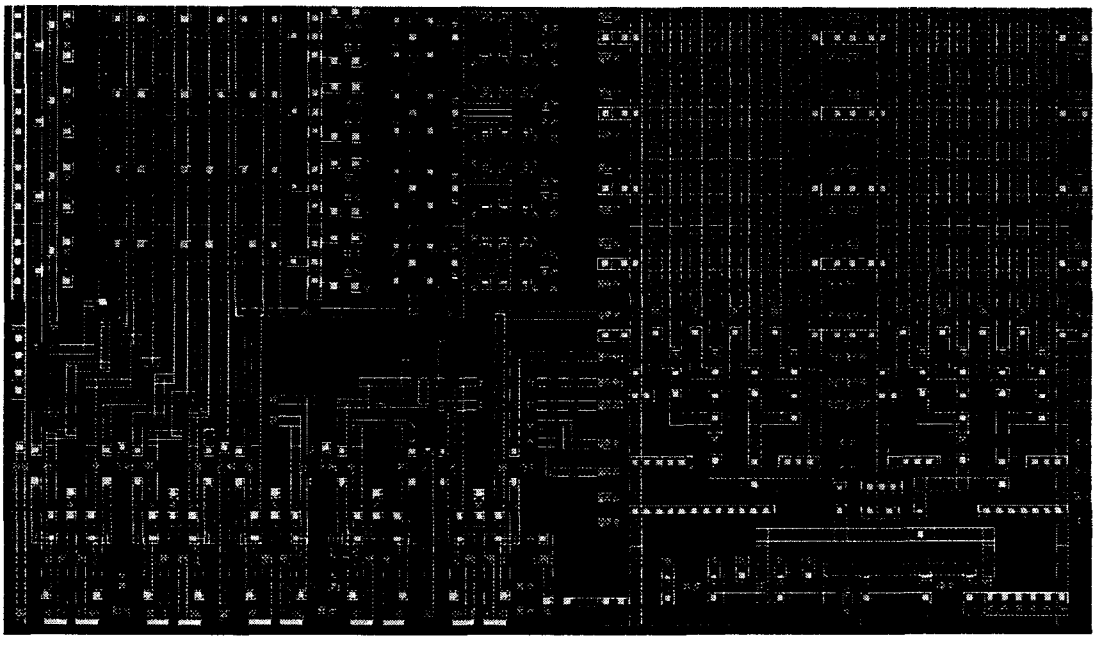


Figura 5.13. Detalle de una ROM de 1024×5 bits.

5.5.6. Resultados experimentales

Con los datos recogidos en los apartados anteriores, se tiene disponible el diseño de una tabla de 1024×5 bits que, tal y como se ha comentado, es la base para la construcción de un sistema de escalado para un RNS definido por tres módulos de cinco bits, con uno de ellos como constante de escalado.

La tabla construida según las especificaciones de los apartados anteriores puede ser simulada para la obtención de los datos relativos a velocidad y consumo de potencia; en concreto, se ha simulado una tabla de 2^{10}×1 bits incluyendo las capacidades y

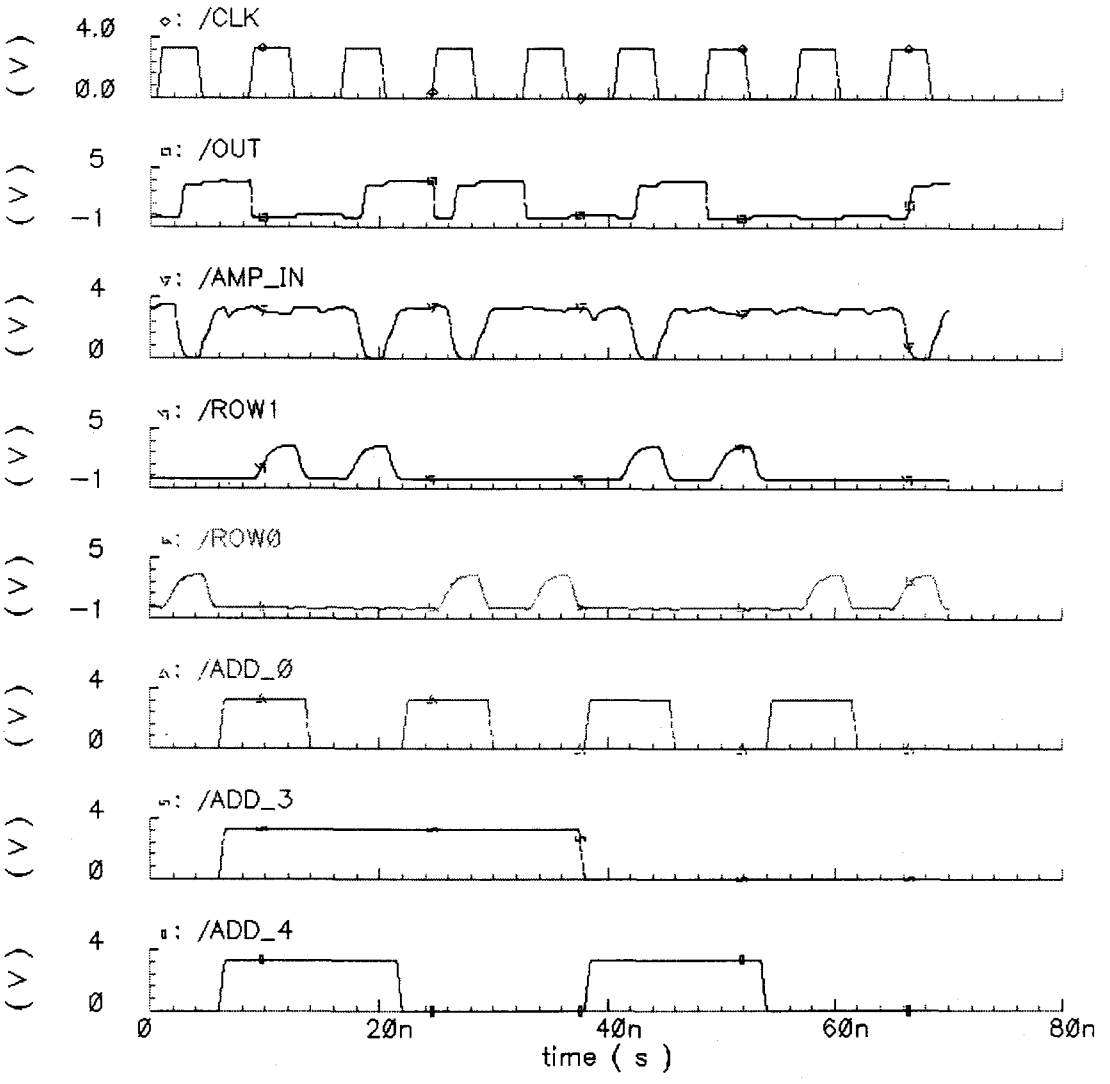


Figura 5.14. Resultados de simulación para una tabla de 2^{10}×1 bits.

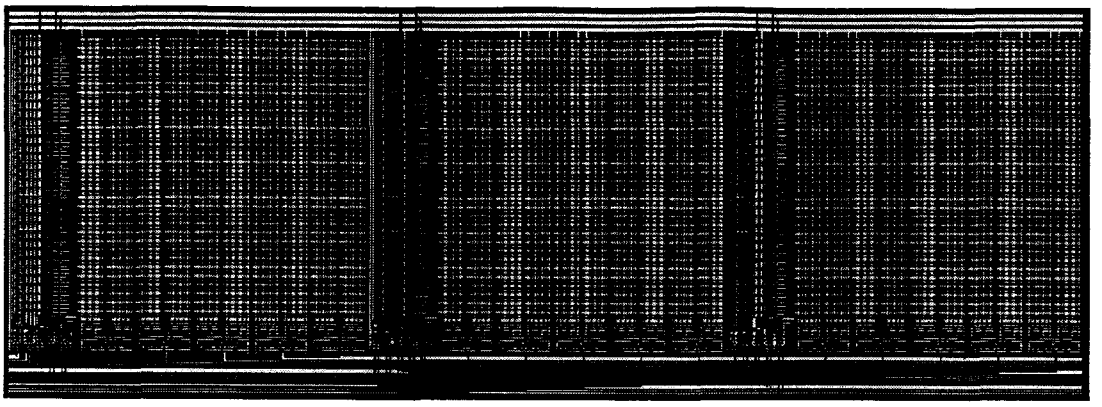


Figura 5.15. Circuito VLSI para el escalado en un RNS de tres módulos de cinco bits.

resistencias de línea asociadas a la tabla de $2^{10}\times5$ bits. Por tanto, los resultados de consumo de potencia y velocidad de funcionamiento para la tabla de $2^{10}\times5$ bits pueden acotarse a partir de los de esta tabla de $2^{10}\times1$ bits. La figura 5.14 muestra los valores de tensión en diferentes puntos del circuito obtenidos por simulación con spectreS; en la figura se pueden observar las señales generadas para la selección de filas, la tensión en el nodo de precarga de la salida (AMP_IN en la figura) y a tensión de salida. De la simulación se deduce:

- los tiempos de transición máximos son 0.3 ns para escribir 0 en la salida, y **2.3 ns** para generar 1; con esto, la velocidad máxima de funcionamiento de la memoria se sitúa alrededor de los 200 MHz;

- el consumo de potencia para la tabla de 1024×5 bits es de **4.26 mW** para una frecuencia de reloj de 125 MHz.

De este modo, es posible diseñar un sistema VLSI para la realización del escalado en un RNS de tres módulos de cinco bits; la estructura, con tres tablas de 1024×5 bits, se muestra en la figura 5.15. En ella, las tres tablas están acompañadas por líneas de distribución de GND, Vdd y CLK, en la parte superior de la misma, mientras que en la parte inferior se sitúan las líneas de datos. Así, si $m_3 \equiv K$, según se vio en la sección 5.4, las dos primeros tablas calculan y_1 e y_2 con $[x_1, x_3]$ y $[x_2, x_3]$ como entradas, respectivamente; finalmente, la tercera tabla genera y_3 a partir de $[y_1, y_2]$. Por tanto, es posible conseguir un sistema de escalado capaz de funcionar a 200 MHz, con cauce segmentado, y en

consecuencia, procesando 200 millones de datos por segundo; esto requiere un área de 1199.6μm×424.7μm, mientras que el consumo de potencia es inferior a 13 mW cuando la frecuencia de reloj es de 125 MHz.

Esta sección ha mostrado que es posible la aplicación de los algoritmos creados en circuitos ASIC VLSI. En concreto, se ha visto cómo el escalado para un RNS definido por tres módulos de cinco bits con uno de ellos como constante de escalado puede realizarse en 0.509 mm^2, a velocidades de procesamiento muy elevadas y con un reducido consumo de potencia.

5.6. CONCLUSIÓN

Este capítulo ha mostrado algunos de los algoritmos clásicos para la realización del escalado, al mismo tiempo que se han propuesto mejoras para éstos y se han mostrado nuevos algoritmos y posibles aplicaciones de los mismos; podemos, pues, resumir las aportaciones mostradas en este capítulo en los siguientes puntos:

- Se han desarrollado las bases para la creación de un sistema de propósito general para el RNS, con la creación de un sistema de escalado basado en módulos aritméticos convencionales; en principio, puede que esta alternativa no parezca ventajosa, pero la inclusión de segmentación de cauce en ella conduce a mejores resultados que la aplicación directa de las propuestas previas.

- Dado que los algoritmos clásicos para el escalado están basados en el uso de tablas de consulta, se ha mostrado un nuevo algoritmo de escalado basado en un método de consulta pura, ya que todos los resultados están perfectamente definidos por un cierto conjunto de residuos; así, el esquema de escalado propuesto tiene un tiempo de cálculo de sólo dos ciclos de acceso a memoria, independientemente del número de módulos del RNS empleado o el número de módulos presentes en la constante de escalado.

- Aunque el esquema de escalado anterior conduce a un tiempo de cálculo independiente del número de módulos y de K, puede resultar muy costoso en términos de memoria; por tanto, se han propuesto dos nuevos algoritmos que combinan los anteriores métodos iterativos con el nuevo método de consulta pura, consiguiendo un compromiso entre velocidad y coste;

- Finalmente, se ha presentado un ejemplo de diseño VLSI para el algoritmo de escalado basado en tablas de consulta introducido en este capítulo; en concreto, se ha utilizado un proceso CMOS de 0.6μm para diseñar un circuito para el escalado en un RNS definido por tres módulos de cinco bits, con uno de ellos como constante de escalado, y se ha conseguido un sistema compacto, de elevada velocidad y bajo consumo de potencia.

CAPÍTULO VI

DESARROLLO DE APLICACIONES RNS SOBRE DISPOSITIVOS PROGRAMABLES

En los capítulos previos se han desarrollado nuevas propuestas para diferentes circuitos aritméticos en el RNS, cubriendo las operaciones básicas: suma y resta, multiplicación y escalado. En este capítulo se van a aplicar estos conceptos a la creación de aplicaciones reales utilizando dispositivos programables; en concreto, se desarrollará una estructura para la implementación de los filtros CIC (Cascade Integrator Comb) o de Hogenauer [HOG81]; estos filtros son especialmente atractivos ya que sólo requieren el uso de sumadores. Por otra parte, se presentará una nueva metodología para la implementación de filtros FIR basados en el RNS usando la aritmética distribuida (DA: Distributed Arithmetic) [WHI89]. Se verá que el RNS conduce a aplicaciones más ventajosas que la aritmética binaria en complemento a dos en realizaciones sobre CPLDs.

6.1. INTRODUCCIÓN

Como ya se ha comentado, en la actualidad se disponen de varias vías [GAR98a] para el desarrollo de aplicaciones de procesamiento digital de señales con circuitos integrados de aplicación específica (ASIC: Application Specific Integrated Circuit): circuitos VLSI diseñados a nivel de transistor, celdas estándares y matrices de puertas; por otra parte, también se dispone de los circuitos programables (FPGAs y CPLDs); y por

último, microprocesadores especialmente diseñados para aplicaciones de procesamiento digital de señales, también conocidos genéricamente como DSPs. En los últimos tiempos los circuitos ASIC se han usado como una alternativa a los DSPs, pero en la actualidad los circuitos programables están abriendo un nuevo campo frente a estos microprocesadores especiales. Como ejemplo, un microprocesador TMS320C60 a 200 MHz con dos multiplicadores especiales en su interior puede realizar un filtro FIR de 16 etapas con una velocidad de procesamiento de 25 millones de muestras por segundo o MSPS (MegaSamples Per Second), mientras que un dispositivo programable puede implementar el mismo filtro a más de 100 MSPS [GAR98a]. Además, la arquitectura de los dispositivos programables, con un biestable a la salida de cada bloque lógico, permite una fácil implementación de la segmentación de cauce sin necesidad de emplear recursos adicionales.

Sin embargo, hasta el momento, el uso de dispositivos programables ha encontrado limitaciones en aplicaciones de gran ancho de banda y precisión elevada, debido al considerable incremento de los recursos necesarios para mantener la velocidad de las operaciones aritméticas a realizar. En este capítulo mostraremos cómo el RNS es una alternativa eficiente para solventar estas limitaciones, ya que permite la creación de aplicaciones con mayor ancho de banda que los equivalentes binarios, conservando los recursos requeridos en márgenes razonables.

6.2. DESARROLLO DE FILTROS CIC PARA EL RNS

La primera aplicación que se va a mostrar como ejemplo de las ventajas del uso del RNS sobre dispositivos programables es el desarrollo de filtros CIC, con un filtro CIC de tres etapas con entrada de 8 bits, D=2, R=32 y salida de 10 bits como ejemplo de diseño. Esta estructura basada en el RNS será comparada con la implementación con aritmética binaria convencional en complemento a dos.

6.2.1. Los filtros CIC de Hogenauer

Este tipo de filtros [HOG81] ha demostrado ser de gran utilidad en sistemas con interpolación o supresión (*decimation*). Una de las características más interesantes de los filtros CIC (Cascade Integrator Comb) es que proporcionan filtros FIR empleando sólo sumadores y elementos de retardo; aunque la respuesta en frecuencia fuera de la banda de paso no es suficiente para algunas aplicaciones, filtros convencionales de baja velocidad pueden corregir esto.

La función de transferencia de un sistema CIC de S etapas referida a la frecuencia de muestreo superior viene dada por:

$$H(z) = \left(\frac{1 - z^{-RD}}{1 - z^{-1}} \right)^S = \left(\sum_{k=0}^{RD-1} z^{-k} \right)^S \qquad \textbf{(6.1)}$$

La figura 6.1 muestra un filtro CIC de tres etapas para supresión; éste consta de tres integradores en cascada, una reducción de la frecuencia de muestreo en un factor R y tres filtros peine; la estructura del filtro para interpolación es equivalente, salvo que las etapas peine se anteponen a la cascada de integradores, aunque éstos siguen funcionando a una frecuencia de muestreo superior en un factor R a la de las etapas peine. De la ecuación (6.1) se deduce que S polos a frecuencia cero son anulados por S ceros. El resultado es que la función de transferencia es la misma que la de un filtro de media desplazada (*moving*

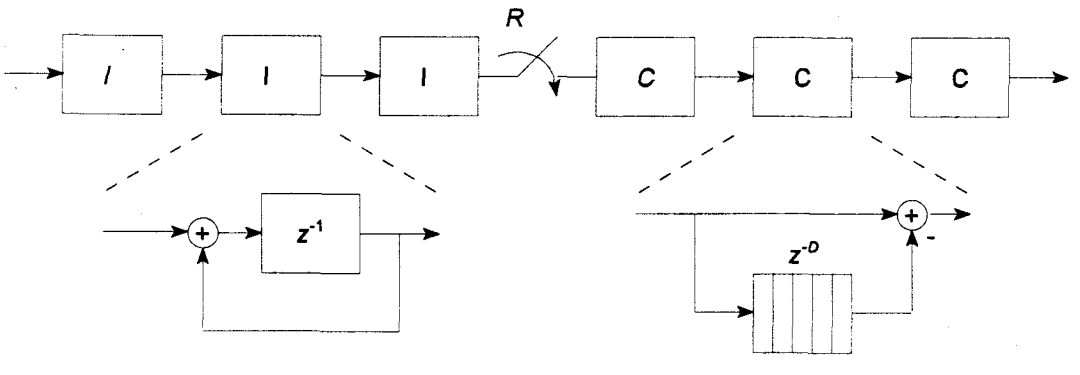


Figura 6.1. Filtro CIC de tres etapas.

average) de S etapas. De este modo, un sistema CIC equivale a la sucesión de S filtros FIR idénticos, con lo que la ventaja de la arquitectura CIC es evidente: no se requieren multiplicadores.

Para un diseño efectivo de la estructura CIC es necesario calcular el máximo crecimiento de los datos. Se puede demostrar que el máximo crecimiento en el rango dinámico de los datos procesados ocurre para frecuencia cero (z=1) y su valor es:

$$B_{crec} = (RD)^S \tag{6.2}$$

que en bits toma el valor $b = \log_2 B_{crec}$.

Para aplicaciones convencionales, la ecuación (6.2) puede conducir a valores bastante elevados, tal y como ocurre en el chip HSP43220 de Harris Semiconductor, que incorpora una estructura CIC de cinco etapas y D=1 con un rango dinámico de 56 bits. Este elevado rango dinámico hace pensar en una posible implementación de la estructura de la figura 6.1 basada en el RNS. En concreto, a lo largo de esta sección vamos a examinar los datos para una estructura CIC de tres etapas con entrada de 8 bits, D=2 y R=32. De este modo, requeriría un rango dinámico en bits de W=8+b, estando b dado por la ecuación (6.2); por tanto, W=8+$\log_2(2 \cdot 32)$=26 bits asegura que no se producirá desbordamiento durante el procesamiento; además, la salida puede limitarse a 10 bits.

Por otra parte, Hogenauer [HOG81] llegó, tras un riguroso análisis, a la conclusión de que la estructura CIC presenta un comportamiento muy favorable frente al redondeo y el truncamiento de los datos; así, los bits menos significativos de algunas etapas podrían ser eliminados sin afectar a la integridad del sistema. En concreto, se puede demostrar [HOG81] que el número B_j de bits menos significativos que se puede despreciar en la etapa j-ésima del filtro (j=1 para el primer integrador, y j=2S para el último peine) es:

$$B_j = \left\lfloor -\log_2 F_j + \log_2 \sigma_T + \frac{1}{6}\log_2 \frac{6}{S} \right\rfloor \tag{6.3}$$

donde σ_T^2 es la varianza del error total por truncamiento y redondeo, y:

$$F_j^2 = \begin{cases} \sum_{k=0}^{(RD-1)S} h_j^2(k) & j = 1, 2, \cdots, 2S \\ 1 & j = 2S+1 \end{cases} \quad \textbf{(6.4)}$$

siendo:

$$h_j(k) = \begin{cases} \sum_{l=0}^{\lfloor k/RD \rfloor} (-1)^l \binom{S}{l} \binom{S-j+k-RDl}{k-RDl} & j = 1, 2, \cdots, S \\ (-1)^k \binom{2S+1-j}{k} & j = S+1, \cdots, 2S \end{cases} \quad \textbf{(6.5)}$$

La figura 6.2 muestra la función de transferencia para la estructura propuesta de tres etapas para dos situaciones diferentes: la línea continua corresponde a que todas las etapas mantegan un rango dinámico de 26 bits; la línea de puntos muestra la respuesta del sistema cuando el rango dinámico es de 26 bits para el primer integrador, 20 para los otros dos integradores y 14 bits para las etapas peine. La diferencia en respuesta es prácticamente inapreciable, con lo que se puede reducir el rango dinámico y, por tanto, el hardware asociado, en algunas de las etapas, sin afectar sustancialmente a la respuesta del sistema.

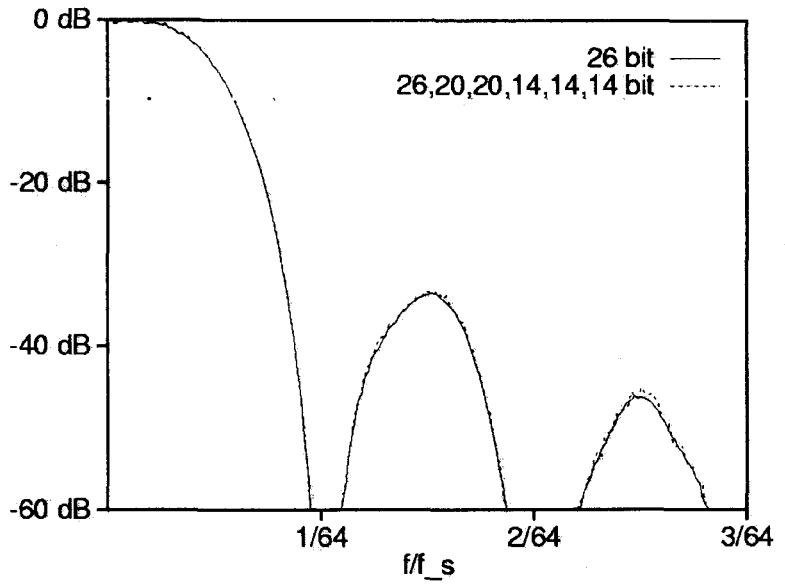


Figura 6.2. Respuesta en frecuencia de un filtro CIC de tres etapas con y sin modificación del rango (f_s es la fecuencia de muestreo en los integradores)

6.2.2. Sumadores RNS para los filtros CIC sobre dispositivos programables

El desarrollo de la estructura de la figura 6.1 requiere el uso de sumadores rápidos, por lo que es preciso elegir la estructura más adecuada para la realización de los sumadores en módulo. Dado que se va a implementar el filtro CIC sobre un dispositivo de grado 4 (el tiempo típico del dispositivo es 4ns) de la familia FLEX10K de Altera (v. apéndice B), de todas las estructuras vistas en la sección 3.1 se han seleccionado las expuestas en el apartado 3.1.1, en concreto el sumador basado en sumadores binarios de la figura 3.1 y el sumador híbrido de la figura 3.2.

Como los dispositivos de la familia FLEX10K constan de bloques lógicos (LAB) de 8 elementos (LE) y cada LE es una tabla de tamaño $2^3 \times 1$ con una cadena de acarreo de alta velocidad (con un retardo total inferior a 0.5ns, mientras que el retardo de la tabla es de unos 3ns) cuando funciona en modo aritmético, es preciso modificar [GAR98a] ligeramente el sumador de la figura 3.1; se supone que éste usa sumadores en complemento a dos, con lo que sólo es preciso observar el acarreo de la segunda etapa, es decir, si $X+Y-m$ es negativo o no, para elegir el resultado correcto para $(X+Y)$ mod m. Sin embargo, para obtener las máximas prestaciones utilizando CPLDs, la opción más ventajosa es la implementación de sumadores para números sin signo. Así, la estructura más adecuada para la implementación del sumador módulo m es la de la figura 6.3, donde k_m es el número de bits necesario para representar datos módulo m. Así, se obtiene un sumador dotado de segmentación de cauce sin coste adicional, ya que la salida de cada LE consta de un biestable; además, si la segmentación de cauce no es posible en alguna aplicación, sólo hay que eliminar los registros correspondientes para obtener una versión no segmentada del sumador. Por otra parte, los chips de la familia FLEX10K constan de tablas de $2^8 \times 8$ bits llamadas EABs (v. apéndice B), que también pueden configurarse como tablas de $2^9 \times 4$, $2^{10} \times 2$ ó $2^{11} \times 1$ bits; esto permite implementar el sumador híbrido de la figura 3.2; sin embargo, para alcanzar la mayor velocidad de funcionamiento de estas tablas es necesario que trabajen en modo síncrono, de manera que dirección y datos de salida se almacenan en sendos registros, lo que permite un funcionamiento segmentado.

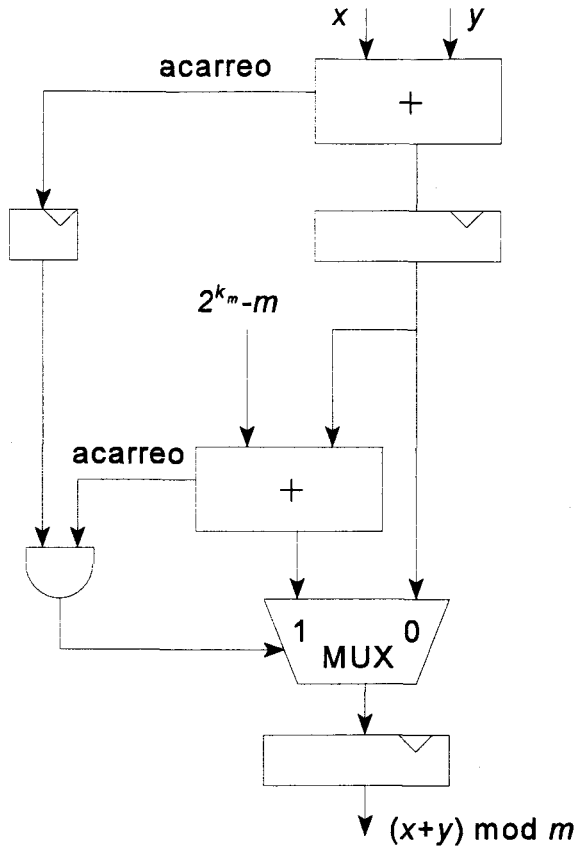


Figura 6.3. Modificación del sumador basado en sumadores binarios para implementación con CPLDs FLEX10K de Altera.

TABLA 6.1

Bits	6	7	8
Sumador figura 6.3 (no segmentado)	41.3 MHz 15 LEs	44.6 MHz 18 LEs	33.6 MHz 19 LEs
Sumador figura 6.3 (segmentado)	76.3 MHz 16 LEs	62.5 MHz 18 LEs	60.9 MHz 20 LEs
Sumador híbrido (segmentado)	70.4 MHz 7 LEs, 1 EAB	70.4 MHz 8 LEs, 1 EAB	60.6 MHz 9 LEs, 2 EABs

La tabla 6.1 recoge los datos de uso de LEs y EABs, así como la velocidad máxima de funcionamiento para estos sumadores con módulos de diferentes anchos de palabra. A pesar de que el sumador híbrido proporciona una alta velocidad, la necesidad de usar los EABs en modo síncrono propicia que estas tablas funcionen con un retardo de cuatro

ciclos en la segmentación de cauce; además, las tablas serán usadas para la realización de otras operaciones, como se mostrará más adelante. De este modo, se ha elegido el sumador de la figura 6.3 con segmentación de cauce para el diseño de un filtro CIC. Por otra parte, es posible extraer algunas conclusiones interesantes de los datos de la tabla 6.1:

- La mayor velocidad con segmentación de cauce se obtiene para módulos de 6 bits, ya que en este caso, cada una de las dos etapas de la figura 6.3 puede ser albergada en un solo LAB (de ahí el uso de 16 LEs para estos sumadores), además de que el tiempo de propagación del acarreo es el menor posible.

- Para módulos de 8 o más bits, ambas etapas han de ser distribuidas en dos LABs, con lo que es necesario propagar las cadenas de acarreo de alta velocidad entre ellos, generándose retardos adicionales.

- Si se elige la alternativa no segmentada para el sumador de la figura 6.3, curiosamente el sumador de 7 bits es más rápido que el de 6; esto se debe a que la primera etapa ocupa un LAB completo, y se puede aprovechar la salida de propagación de la cadena de acarreo del LAB para transmitirlo a la siguiente etapa; cuando se segmenta, el bit de acarreo ha de pasar por un registro, lo que elimina esta ventaja, además de introducir un LE adicional para módulos de 6 y 8 bits.

- Por último, para módulos de 8 bits, la alternativa híbrida necesita una tabla de $2^9 \times 8$ bits; dado que el tamaño máximo del EAB es $2^8 \times 8$, se requieren dos EABs para realizar la corrección en módulo; además del coste adicional, se introduce mayor retardo, al ser esta estructura más lenta que un sólo EAB, que es el factor limitante en velocidad para los otros dos casos.

La latencia del cauce de dos ciclos en el funcionamiento del sumador de la figura 6.3 no supone ningún problema para la implementación de estructuras no recursivas, tales como los filtros peine de la figura 6.1. Sin embargo, si este sumador con cauce segmentado es utilizado como acumulador, utilizando su salida como una de las entradas como en el caso de los integradores de la arquitectura CIC, se produce un comportamiento

incorrecto; en concreto, la función de transferencia del sumador de la figura 6.3 cuando se utiliza como acumulador es:

$$F(z) = \frac{z^{-2}}{1 - z^{-2}} \tag{6.6}$$

La ecuación (6.6) difiere claramente de la respuesta deseada para un integrador, ya que la inclusión de estos sumadores en un integrador o acumulador introduce un segundo polo en la mitad de la frecuencia de muestreo, es decir, en π. Este polo puede compensarse con una sección peine (ver figura 6.1) de retardo $D=1$; así, el acumulador seguido de esta sección peine que se muestra en la figura 6.4 se comporta realmente como un integrador con una latencia de cuatro ciclos, ya que su función de transferencia es:

$$F'(z) = \frac{z^{-4}}{1 - z^{-1}} \tag{6.7}$$

Sin embargo, en una aplicación CIC de supresión (*decimation*), que es la que se va a implementar, es posible asumir que el filtro en sí introduce suficiente supresión de las componentes de la señal cercanas a π como para que un segundo filtro paso-banda en π, como el que introduce el acumulador recursivo con cauce segmentado y corrección de polo, no aporte efectos adicionales significativos. De este modo, se utilizarán acumuladores sin compensación del polo, aunque en otras aplicaciones donde esto no sea posible será necesario implementar la estructura de la figura 6.4 para la realización de un acumulador módulo *m* empleando el sumador propuesto en este apartado.

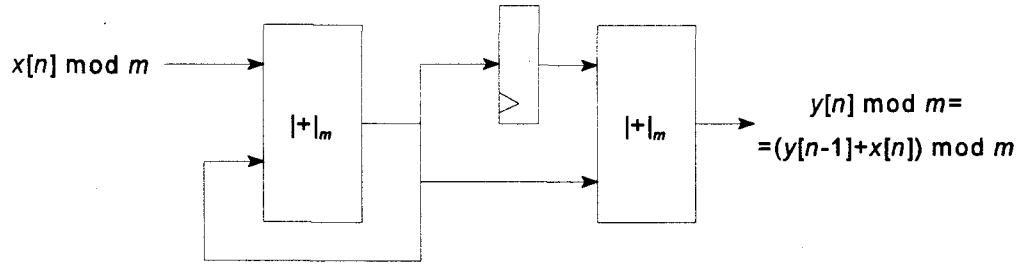


Figura 6.4. Acumulador segmentado módulo *m* con compensación de polo.

6.2.3. Elección de los módulos para el RNS

El ejemplo de diseño es un sistema CIC para supresión de tres etapas (S=3), con entrada de 8 bits, D=2, R=32 y salida de 10 bits; tal y como se ha comentado y de acuerdo con lo visto en el apartado 6.2.1, la aplicación a realizar requiere un rango dinámico total de 26 bits.

Para una implementación RNS hay que elegir un conjunto adecuado de módulos que satisfagan el rango dinámico exigido, o escalar entre diversas etapas. En este caso, dado que el rango no es excesivo, se optará por un conjunto de módulos que cubra el rango dinámico; de este modo, tres módulos de 6 bits y uno de 8 son suficientes. Puesto que la velocidad de operación del sistema vendrá dada por el módulo más lento, y en virtud de los datos de la tabla 6.1, no parece razonable incluir un módulo de 8 bits en el RNS a emplear. Sin embargo, es posible hacerlo si se elige este módulo como $256=2^8$; de este modo, no es necesario hacer uso de la estructura de la figura 6.3 y se pueden emplear sumadores binarios convencionales e introducir retardos adicionales en la estructura de la figura 6.1 para el canal módulo 256 a fin de sincronizar éste con el resto de canales. Para la familia FLEX10K, los sumadores binarios convencionales requieren un LE por cada bit; además, para dispositivos de grado 4, la velocidad máxima de operación para sumadores de 8 bits incluidos en estructuras segmentadas es de 136.9 MHz, muy superior a la velocidad para los sumadores de la tabla 6.1. Por tanto, se empleará 256 como uno de los módulos, incluyendo en el canal correspondiente a este módulo los elementos de retardo necesarios para igualar el mayor retardo de los demás sumadores y mantener la sincronización de la segmentación de cauce del sistema total.

De este modo, el RNS elegido será el formado por {256, 63, 61, 59}; además, la combinación de estos módulos permite llevar a cabo la reducción del rango dinámico indicada en la figura 6.2. Así, sólo el primer integrador haría uso de estos cuatro módulos, con un rango de 26 bits, mientras los otros dos integradores prescindirían de uno de los módulos de 6 bits, conservando un rango dinámico de 20 bits; por último, las tres etapas peine sólo emplearían en RNS formado por 256 y uno de los módulos de 6 bits, con un

rango dinámico de 14 bits.

6.2.4. Reducción eficiente del rango dinámico

Se ha visto cómo la arquitectura CIC permite la reducción del rango dinámico a lo largo de las etapas sin alterar la respuesta del filtro; más aún, en el apartado anterior se ha mostrado que es posible seleccionar los módulos del RNS para satisfacer las condiciones de la figura 6.2. Sin embargo, esta reducción no es sencilla (en un sistema binario convencional, sólo habría que prescindir de los bits menos significativos) y está relacionada con el problema del escalado expuesto en el capítulo 5. Se vio que la operación de escalado, en virtud de (5.5), se divide en dos operaciones independientes cuando la constante de escalado K es el producto de varios de los módulos: el cálculo del residuo del resultado en los módulos que no forman K y la generación de los residuos correspondientes a los módulos que forman K. Como el objetivo es eliminar un módulo del RNS empleado, se puede considerar alguno de los algoritmos descritos en el capítulo V para el escalado, suprimiendo la etapa final de extensión de base. Este nuevo esquema se llamará *escalado con eliminación de base* o BRS (Base Removal Scaling) [GAR98a]; con esto, no sólo se reduce cada dato al nuevo rango dinámico, sino que es posible continuar el cálculo en un nuevo RNS en el que se elimina el módulo usado como constante de escalado K.

Dado el tamaño de las tablas disponibles en la familia FLEX10K y el hecho de que se pueden conseguir tablas de mayor capacidad configurando varios EABs con las mismas direcciones, pero a costa de reducir la velocidad de funcionamiento total de la tabla (v. tabla 6.1 para sumadores híbridos), sólo se van a emplear tablas de una entrada; de este modo, se hará uso del *algoritmo original* en [JUL78], pero transformando cada tabla de dos entradas en un sumador y una tabla de una entrada que lleva a cabo la multiplicación por un factor constante, que es el inverso multiplicativo de uno de los módulos, tal y como se ha propuesto en la sección 5.3 [GAR98b]. Dado que la velocidad de las memorias es ligeramente inferior (v. tabla 6.1) a la de los sumadores, la inclusión del BRS reducirá la velocidad si se compara con la opción de un rango dinámico de 26 bits en todas las etapas.

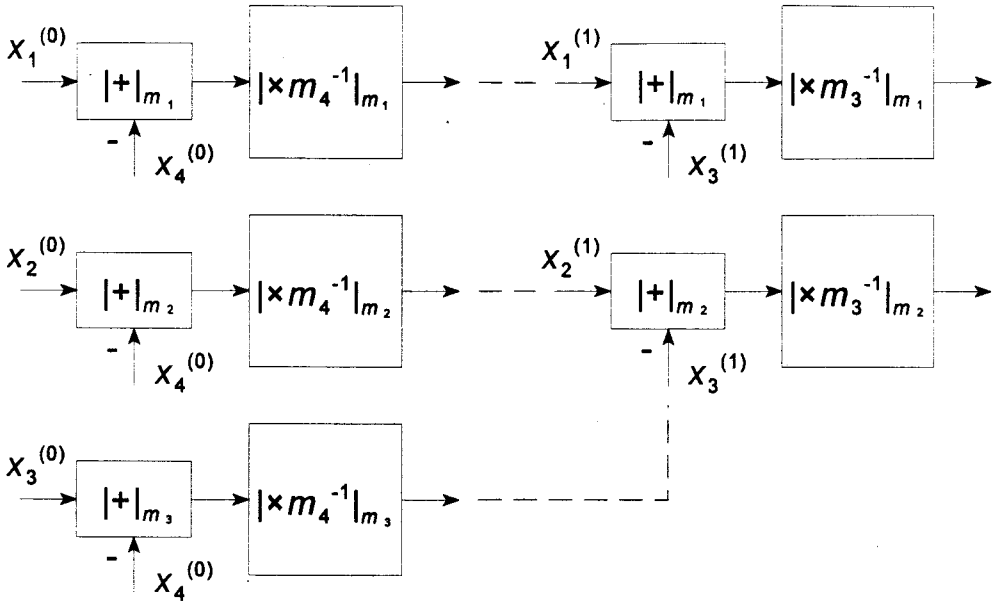


Figura 6.5. BRS para el filtro CIC.

La figura 6.5 muestra la estructura para el BRS en el ejemplo propuesto, con la supresión de dos módulos de 6 bits, m_3 y m_4; para que este esquema sea válido, sólo se han de elegir los módulos de forma que $\{m_1, m_2\}>m_3>m_4$, con lo que con la elección de módulos del apartado anterior es posible realizar la asignación m_1=256, m_2=63, m_3=61 y m_4=59. Los superíndices en la figura indican la etapa BRS que se lleva a cabo.

6.2.5. Esquemas de conversión eficientes

Los datos en la arquitectura CIC propuesta son de 8 bits, y la salida de 10. No se va a tratar aquí cómo proporcionar datos de 8 bits en el RNS propuesto, ya que (v. apartado 2.1.3) con este rango dinámico de entrada y este tamaño de los módulos, se pueden crear conversores A/D que proporcionen una salida adecuada para cada módulo.

Con respecto a la conversión de la salida a un entero binario de 10 bits, se vio en el apartado 2.1.3 cómo el CRT proporciona el algoritmo adecuado para la conversión desde el RNS a binario a través de la ecuación (2.8). Sin embargo, el RNS elegido define

un rango de 26 bits, con lo que la ecuación (2.8) llevaría a calcular un entero de 26 bits para despreciar finalmente los bits menos significativos, lo que parece no tener mucho sentido; por otra parte, incluso con la reducción detallada de rango dinámico a través de dos etapas de BRS, la etapa final define un rango de 14 bits. De este modo, sería necesario realizar una costosa implementación CRT calculando bits que se van a despreciar, o bien escalar el resultado final antes de su conversión a través del CRT; esto último tampoco parece una buena opción, dado que los algoritmos más eficientes para el escalado necesitan que la constante de escalado corresponda a uno o varios de los módulos; con todo ello, se podría reducir la salida a un rango de 8 bits, pero en este caso se perdería precisión en la salida. Existe una solución, propuesta por Griffin, Sousa y Taylor [GRI88, GRI89], llamada ε-CRT, que lleva a cabo la conversión a binario y el escalado en un sólo paso; este método es muy sencillo y transforma la ecuación (2.8) de una conversión entre la n-tupla de residuos y un valor módulo M a un escalado con constante de escalado V, siendo V un número real, y una conversión de la representación en residuos a un valor módulo $\lfloor M/V \rfloor$. En concreto, se puede demostrar [GRI88] que la versión escalada de la ecuación (2.8) es:

$$X^S \equiv \left\lfloor \frac{X}{V} \right\rfloor = \left(\sum_{i=1}^{n} \left\lfloor \phi(x_i)/V \right\rfloor \right) \bmod \lfloor M/V \rfloor \qquad \textbf{(6.8)}$$

donde:

$$\phi(x_i) = s_i \left(x_i s_i^{-1} \right) \bmod m_i \qquad \textbf{(6.9)}$$

Las ecuaciones (6.8) y (6.9) conducen a un esquema sencillo de conversión, ya que:

- se pueden precalcular los valores de $\lfloor \phi(x_i)/V \rfloor$ y almacenarlos en una tabla de consulta; de este modo, sólo son necesarias n tablas para generar los n términos de la ecuación (6.8); cada una de estas tablas es de $2^{k_i} \times k_{\lfloor M/V \rfloor}$ bits, siendo k_i el número de bits necesarios para representar valores módulo m_i, y $k_{\lfloor M/V \rfloor}$ el número de bits necesarios para representar valores módulo $\lfloor M/V \rfloor$;

- por tanto, sólo se requiere un acumulador módulo $\lfloor M/V \rfloor$ para finalizar el cálculo de (6.8); esto supone una gran ventaja frente al algoritmo CRT tradicional de la

ecuación (2.8), al pasar de un acumulador de k_M bits a uno de $k_{\lfloor M/V \rfloor}$ bits. Además, dado que V puede ser cualquier valor real, sólo hay que elegir V de forma que $\lfloor M/V \rfloor$ sea una potencia de dos, lo que simplifica de manera sustancial el problema.

La gran ventaja del ε-CRT frente al CRT convencional, además de realizar el escalado al mismo tiempo que la conversión de RNS a binario, es la sustitución de un acumulador módulo M por sumadores convencionales para implementar un acumulador módulo una potencia de dos. Por otra parte, dado que en la ecuación (6.8) se utiliza la parte entera de (6.9) en lugar de su valor exacto, el ε-CRT, a través de la ecuación (6.8), introduce un pequeño error al acercarnos al límite del rango dinámico, M, con respecto de la ecuación (2.8); sin embargo, esto no constituye inconveniente alguno si consideramos que el rango dinámico seleccionado ha de ser suficiente para no ser rebasado (o se realizarán las operaciones de escalado necesarias para evitarlo).

6.2.6. Resultados experimentales

Recopilando lo indicado en los apartados 6.2.4 y 6.2.5, tres son las posibles implementaciones basadas en el RNS de la estructura CIC de la figura 6.1 con entrada de 8 bits, D=2, R=32 y salida de 10 bits:

1. Implementar todas las etapas de la arquitectura CIC con un rango dinámico de 26 bits y realizar una etapa final de ε-CRT con una conversión desde el RNS con 26 bits de rango dinámico a binario de 10 bits, para generar la salida.

2. Implementar todas las etapas de la arquitectura CIC con un rango dinámico de 26 bits y realizar la conversión final implementando dos etapas de BRS consecutivas siguiendo el esquema de la figura 6.5, para pasar de 26 a 14 bits de rango dinámico y, posteriormente, realizar una etapa ε-CRT para la conversión final para generar la salida de 10 bits; de este modo, el ε-CRT sólo incluiría dos de los módulos y, por tanto, sólo dos tablas; más aún, las dos últimas tablas de la figura 6.5 pueden ser modificadas para incluir el contenido de las correspondientes al ε-CRT; de este

modo, podemos conseguir un ahorro en el número de tablas.

3. Implementar dos etapas BRS a lo largo del filtro, para conseguir los rangos dinámicos en cada etapa mostrados en la figura 6.2 y realizar la generación de la salida a través del ε-CRT con sólo dos módulos, tal y como muestra la figura 6.6.

Los datos de velocidad máxima y coste para los tres esquemas de escalado propuestos están reflejados en la tabla 6.2, a partir de las síntesis VHDL [CHA97] de los mismos. Hay que notar que para las dos primeras alternativas, la velocidad es inferior a la de los sumadores, pero dado que el escalado y conversión finales se realizan con un factor de reducción de velocidad R=32, pueden funcionar con un reloj 32 veces más lento que el de las etapas integradoras y este menor rendimiento no es importante en este caso; del mismo modo, para la última alternativa, tal como muestra la figura 6.6, sólo la primera etapa BRS funciona a la velocidad de las etapas integradoras, con lo que se muestra la velocidad máxima para esta etapa de BRS de m_4. Respecto a las dos primeras alternativas, la elección entre una u otra dependerá del factor limitante en el chip a

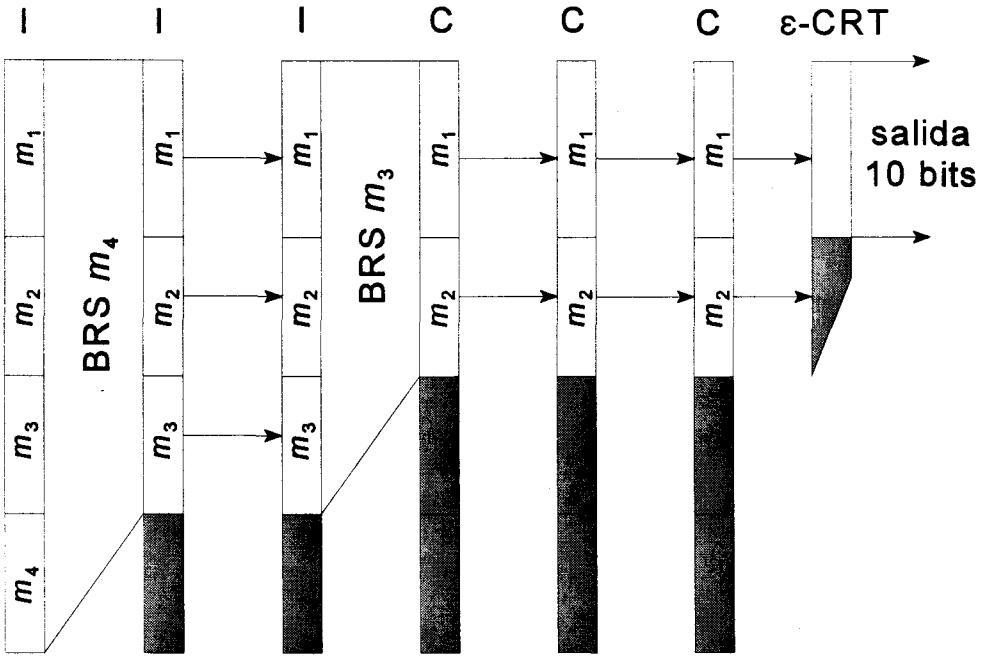


Figura 6.6. Diseño con rango dinámico detallado del filtro CIC.

emplear: elementos lógicos o tablas de memoria.

TABLA 6.2

	ε-CRT (1)	BRS + ε-CRT (2)	BRS y ε-CRT (3)
Velocidad (MHz)	58.8	58.8	70.4
Coste	34 LEs, 8 EABs	87 LEs, 7 EABs	87 LEs, 9 EABs

Por otra parte, la tabla 6.3 recoge los resultados para lo que es en sí la arquitectura CIC: los integradores y secciones peine de la figura 6.1. Esta tabla no incluye datos de coste acerca de escalado o conversión, y no habría más que combinar ambas tablas para obtener los datos relativos a una implementación concreta del sistema. La velocidad se da en MSPS, que en este caso coincidiría con la máxima frecuencia de reloj del sistema, al proporcionar el cauce segmentado un dato en cada ciclo de reloj. La tabla 6.3 compara la realización binaria convencional en complemento a dos con palabras de 26 bits con las diferentes propuestas para el RNS. Como se puede apreciar, al incluir las etapas de BRS, tal y como muestra la figura 6.6, en la implementación basada en RNS, la menor velocidad de las tablas, 70.4 MSPS, reduce las prestaciones del sistema.

TABLA 6.3

	Complemento a 2 26 bits	**RNS {256, 63, 61, 59}**	**RNS (rango dinámico detallado)**
Velocidad (MSPS)	49.3	76.3	70.4
Coste (LE)	343	559	355

En cualquier caso, la implementación RNS con 26 bits de rango dinámico en todas las etapas consigue una mejora en velocidad cercana al 54% respecto a la implementación binaria en complemento a dos; por otra parte, la inclusión de la reducción del rango dinámico en la implementación RNS supone un ahorro del 36% con respecto a la alternativa anterior, al tiempo que se mantiene una ventaja en velocidad del 43% sobre la implementación binaria convencional.

6.3. FILTROS FIR PARA EL RNS USANDO ARITMÉTICA DISTRIBUIDA

En esta sección se mostrará que el RNS puede también ser útil para mejorar una de las arquitecturas más populares para la optimización de filtros FIR y otras aplicaciones similares en aritmética binaria.

6.3.1. Aritmética distribuida

La aritmética distribuida [WHI89] (DA: Distributed Arithmetic) recibe este nombre porque las operaciones usuales en procesamiento digital de señales, es decir, suma y multiplicación, no aparecen como tales y quedan, en cierto modo, distribuidas entre diversos elementos. Esta técnica fue usada por primera vez por Peled y Liu [PEL74] y desde su aparición se ha empleado en un gran número de aplicaciones [WHI89] para reducir los recursos requeridos para la realización de sumas de productos: filtrado FIR, sistemas de control, etc. La aritmética distribuida lleva a cabo un cálculo bit a bit (*bit-serial*) de un producto interno de dos vectores con una estructura muy eficiente. Sea la siguiente suma de productos:

$$y = \sum_{k=1}^{K} A_k x_k \tag{6.10}$$

donde A_k son coeficientes constantes y $\{x_k\}$ son los datos de entrada. Supóngase que cada x_k es un número binario de N bits representado en complemento a dos y que está escalado convenientemente, de manera que $|x_k|<1$; de este modo:

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \qquad k = 1, \cdots, K \tag{6.11}$$

siendo b_{k0} el bit de signo y $b_{k,N-1}$ el bit menos significativo. Introduciendo (6.11) en (6.10) se llega a la expresión:

$$y = \sum_{k=1}^{K} A_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right] \tag{6.12}$$

Es evidente que la ecuación (6.12) puede manipularse fácilmente para obtener:

$$y = \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} - \sum_{k=1}^{K} A_k b_{k0} \tag{6.13}$$

que define la aritmética distribuida, dado que el término entre corchetes:

$$\sum_{k=1}^{K} A_k b_{kn} \tag{6.14}$$

sólo puede tomar 2^K valores, ya que b_{kn} toma valores 0 y 1. Así, los 2^K valores de (6.14) pueden precalcularse y almacenarse en una ROM que actuará posteriormente como tabla de consulta; esta tabla se direcciona en la iteración j-ésima de (6.13) por el vector de bits j-ésimos de las entradas, $[b_{1j}, b_{2j}, ..., b_{Kj}]$ y su salida, tras introducir convenientemente el factor 2^{-j}, puede acumularse con el resultado previo. Finalmente, tras las N iteraciones de (6.13) el acumulador contiene el mismo resultado que (6.10) y (6.12), que necesitan K iteraciones.

La implementación de (6.10) requiere un acumulador y K productos, obteniéndose el resultado tras K iteraciones; por contra, la aritmética distribuida, a partir de (6.13), necesita un acumulador, una tabla de 2^K palabras e incluir de algún modo los factores 2^{-n} de (6.13) para obtener el resultado tras N iteraciones. Por tanto, la opción de la aritmética distribuida no parece demasiado ventajosa, salvo que se tenga en cuenta el siguiente hecho: si se comienza el cálculo de (6.13) por el bit menos significativo de los vectores de entrada, en la iteración j-ésima el resultado (6.14) es dos veces más significativo que el resultado previo; de este modo, el esquema de la figura 6.7, comenzando el cálculo por el LSB de las entradas, es una implementación de la ecuación (6.13), donde la inclusión del factor 2^{-1} en la realimentación del acumulador se realiza con un simple desplazamiento a la derecha del resultado previo y T_s es la señal que indica el momento en el que el bit de signo de las entradas está siendo procesado, con lo que el acumulador debe restar el valor de la tabla al resultado previo. Este mismo esquema puede construirse con un sumador en

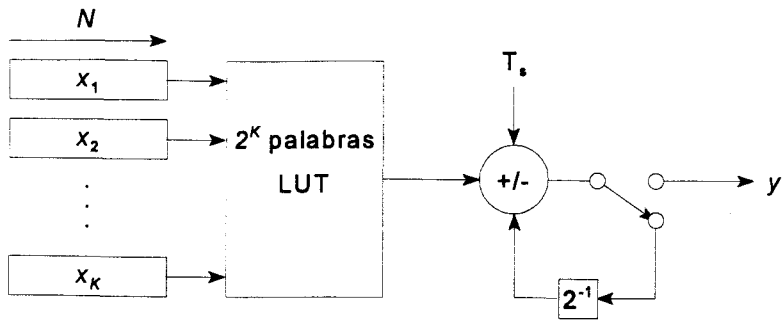


Figura 6.7. Implementación de la aritmética distribuida.

lugar de un sumador/restador si se duplica el tamaño de la tabla para incluir todos los valores de (6.14) y sus correspondientes valores negativos y T_s se aplica como dirección. También existen técnicas [WHI89] para reducir el tamaño de la tabla combinando previamente los bits de dirección. Por tanto, las ventajas de la aritmética distribuida (6.13) quedan de manifiesto con la estructura de la figura 6.7; a pesar de ser una implementación bit a bit de la ecuación (6.10), sólo requiere una tabla de 2^K palabras y un sumador/restador, mientras que la realización directa de (6.10) necesita un sumador y K productos. Lo que es más, son necesarias K iteraciones o ciclos para el cálculo directo de (6.10), mientras que el esquema de la figura 6.7 requiere N iteraciones; de este modo, dependiendo de los valores de N y K, la aritmética distribuida puede resultar más ventajosa en velocidad que la implementación directa.

6.3.2. Traslación de la aritmética distribuida al RNS: DA-RNS

Dado que la aritmética distribuida encuentra su campo de aplicación en la resolución de problemas basados en sumas de productos, es posible pensar en una fusión de la aritmética distribuida y las estructuras basadas en el RNS [GAR99]. De este modo, supóngase un RNS definido por el conjunto de módulos $\{m_1, m_2,..., m_L\}$, con lo que el resultado de la ecuación (6.10) puede expresarse a través de la L-tupla de sus residuos:

$$y_i \equiv y \bmod m_i = \left(\sum_{k=1}^{K} A_k x_k \right) \bmod m_i \equiv \left| \sum_{k=1}^{K} \left| A_k x_{k_i} \right|_{m_i} \right|_{m_i} \qquad \textbf{(6.15)}$$

donde $x_{k_i} = x_k \bmod m_i$. Por tanto, si N_i es el número de bits necesarios para operar módulo m_i, entonces:

$$x_{k_i} = \sum_{n=o}^{N_i-1} b_{k_i,n} 2^n \qquad \begin{matrix} k = 1, \cdots, K \\ i = 1, \cdots, L \end{matrix} \tag{6.16}$$

siendo $b_{k_i,0}$ el bit menos significativo de x_{k_i}, y b_{k_i,N_i-1} el más significativo. Si se compara (6.16) con (6.11), se observa que no existe ninguna diferencia de signo entre los diferentes bits, lo que va a simplificar, como se verá más adelante, la adaptación de la aritmética distribuida al RNS. Introduciendo (6.16) en (6.15):

$$y_i \equiv \left| \sum_{k=1}^{K} \left| A_k \left[\sum_{n=o}^{N_i-1} b_{k_i,n} 2^n \right] \right|_{m_i} \right|_{m_i} = \left| \sum_{n=o}^{N_i-1} \left[\left| \sum_{k=1}^{K} A_k \, b_{k_i,n} \right|_{m_i} \right] 2^n \right|_{m_i} \tag{6.17}$$

que es la ecuación análoga a (6.13). De este modo, definiendo la función:

$$\Phi_i(n) = \left| \sum_{k=1}^{K} A_k \, b_{k_i,n} \right|_{m_i} \equiv \left(\sum_{k=1}^{K} A_k \, b_{k_i,n} \right) \bmod m_i \tag{6.18}$$

se puede escribir la ecuación (6.17) como:

$$y_i \equiv \left| \sum_{n=o}^{N_i-1} \Phi_i(n) \, 2^n \right|_{m_i} \equiv$$

$$\equiv \left(\Phi_i(N_i-1) \, 2^{N_i-1} + \cdots + \Phi_i(1)2 + \Phi_i(0) \right) \bmod m_i \tag{6.19}$$

Por tanto, las ecuaciones (6.17) y (6.19) conducen a un esquema formalmente idéntico a la aritmética distribuida definida por (6.13), sin más que almacenar los valores de la función $\Phi_i(n)$ en una tabla de 2^K palabras para cada canal; de manera similar, es preciso incluir el factor 2^n de (6.19) en el proceso de cálculo, pero es aquí donde el RNS presenta un problema [GAR99]: dado que la ecuación (6.19) ha de calcularse sobre el RNS para obtener una estructura eficiente y aprovechar las ventajas de este sistema, la multiplicación por 2^{-1} de la figura 6.7 no tendría sentido en el RNS; más aún, habría de ser sustituida por la multiplicación por el inverso multiplicativo de 2 para cada módulo, pero esto produciría un resultado incorrecto si el número representado por los L residuos no es múltiplo de 2,

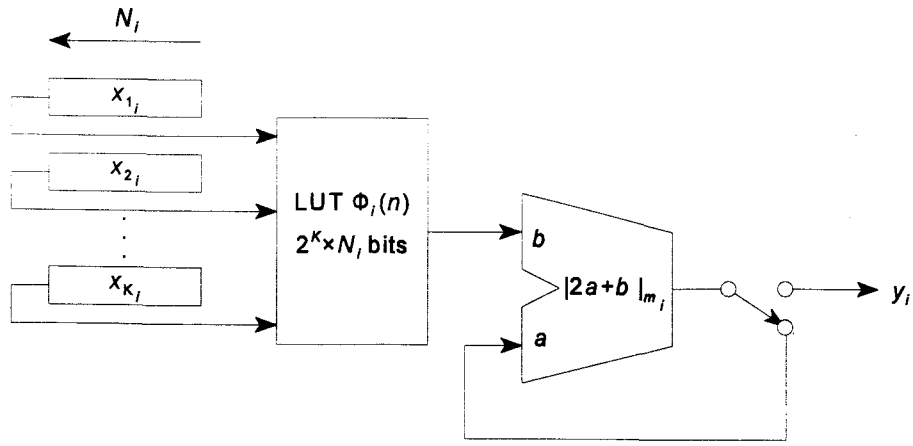


Figura 6.8. Canal módulo m_i de la estructura DA-RNS para la implementación de (6.19).

ya que la división no está definida en el RNS. Una solución consiste en comenzar el cálculo por el bit más significativo de las entradas, de forma que el factor 2^{-1} se convertiría en 2; esto ocurre en virtud de (6.19), ya que al no ser necesario escalar las entradas y no existir potencias negativas en (6.16), todas las potencias que aparecen en (6.19) son positivas. A pesar de esto, la solución tampoco es inmediata, ya que la multiplicación por 2 no se puede realizar en el RNS con un desplazamiento a la izquierda, como en el caso de la aritmética binaria; dado que se trata de un factor constante, es posible implementar esta multiplicación con una tabla de consulta, pero dado que estaría inmersa en un bucle recursivo, no es una solución apropiada. De este modo, se propone la arquitectura [GAR99] de la figura 6.8 para la implementación de la ecuación (6.19), empleando un sumador modificado que calcula $(2a+b)$ mod m_i; en lo que sigue, se denominará a esta estructura DA-RNS. En el siguiente apartado se propondrá una posible estructura para el sumador modificado.

6.3.3. Sumador módulo *m* para DA-RNS

En el apartado 6.2.2 se perfiló el diseño de sumadores para el RNS empleando dispositivos programables. La figura 6.3 muestra la estructura elegida para la implementación de un sumador módulo *m* sobre la familia FLEX10K de Altera. En este

apartado se va a mostrar una estructura similar para la implementación de aplicaciones DA-RNS, con el desarrollo de un sumador para $(2a+b)$ mod m, tal y como recoge el siguiente teorema [GAR99]:

Teorema 6.1: *Sean* $a,b \in Z_m$, *y sea* $N=\lceil \log_2 (m-1) \rceil$ *el número de bits necesario para representar los elementos de* Z_m. *Entonces, la estructura de la figura 6.9 produce como resultado* $(2a+b)$ *mod* m *cuando el control de selección del multiplexor está definido según la tabla 6.4, siendo* C_0 *es el bit más significativo de la primera etapa y* c_1, c_2 *y* c_3 *los acarreos de la primera, segunda y tercera etapas, respectivamente.*

TABLA 6.4

c_1	C_0	c_2	c_3	$(2a+b)$ *mod* m
0	0	0	0	S1
0	0	0	1	S1
0	0	1	0	S2
0	1	0	0	S2
0	1	0	1	S3
0	1	1	0	S3
1	0	0	0	S3
cualquier otra combinación				nunca ocurre

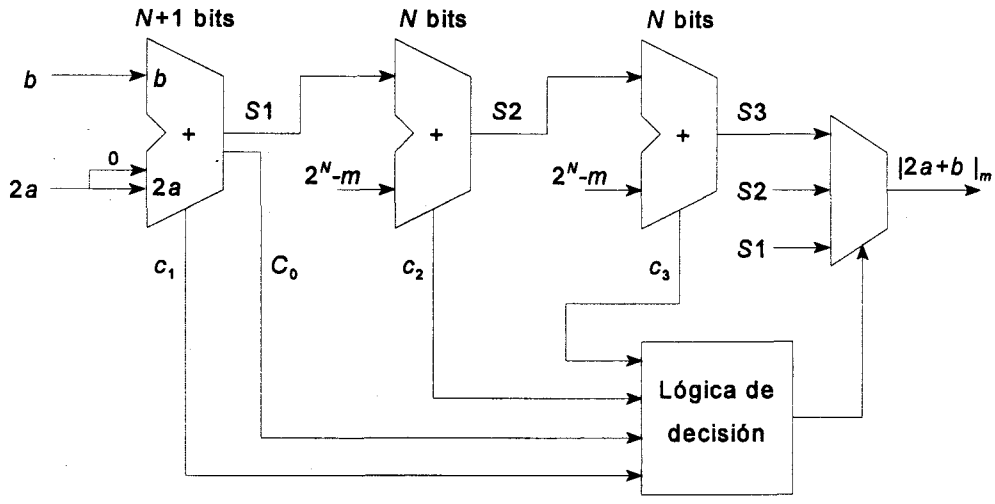


Figura 6.9. Sumador módulo m para calcular $(2a+b)$ mod m.

Demostración: es directa y se reduce a estudiar tres posibles casos:

1. $2a+b<m$: en este caso es claro que el resultado correcto es $S1$; para el primer sumador:

$$2a + b < m < 2^N$$

y c_1C_0=**00**, mientras que para la segunda etapa:

$$(2a + b) + 2^N - m < m + 2^N - m = 2^N$$

con lo que c_2=**0**. Por tanto, se ve que en este caso la salida ha de ser $S1$ y, para ello, ha de satisfacerse $c_1C_0c_2$=**000**, como indica la tabla 6.4.

2. $m \le 2a+b<2m-1$: bajo este supuesto, el resultado correcto es el de la segunda etapa, $S2$. Es preciso distinguir varias situaciones para este caso:

- si $2a+b<2^N$, al igual que para el caso anterior, es claro que c_1C_0=**00** y para la segunda etapa:

$$(2a + b) + 2^N - m < 2m + 2^N - m = 2^N + m < 2^{N+1}$$

$$2a + b + 2^N - m \ge m + 2^N - m = 2^N$$

y es obvio que:

$$2^N \le (2a + b) + 2^N - m < 2^{N+1}$$

de forma que c_2=**1**. Además, la entrada a la tercera etapa es $(2a+b+2^N-m)-2^N$; de este modo, en la tercera etapa se calcula:

$$(2a + b + 2^N - m) - 2^N + 2^N - m = 2a + b + 2^N - 2m < 2m + 2^N - 2m = 2^N$$

y claramente c_3=**0**. Por tanto, en este primer caso con salida $S2$, la combinación de acarreos correcta es $c_1C_0c_2c_3$=**0010**, que corresponde a uno de los casos de la tabla

6.4 con salida $S2$;

- si $2^N \leq 2a+b<2^{N+1}$, entonces $\boldsymbol{c_1C_0=01}$ y la entrada a la segunda etapa es $(2a+b)-2^N$, con lo que el segundo sumador calcula:

$$(2a + b - 2^N) + 2^N - m = 2a + b - m < 2m - m = m < 2^N$$

De este modo, $\boldsymbol{c_2=0}$ y $2a+b-m$ es la entrada del tercer sumador, en el que:

$$(2a + b - m) + 2^N - m = 2a + b + 2^N - 2m < 2m + 2^N - 2m = 2^N$$

con lo que $\boldsymbol{c_3=0}$. Reuniendo todos los resultados, la segunda combinación que produce salida $S2$ es $\boldsymbol{c_1C_0c_2c_3=0100}$, que corresponde al otro caso de la tabla 6.4 con salida $S2$.

3. $2m-1 \leq 2a+b<3m-2$: para esta última situación, el resultado correcto es $S3$. De nuevo se hace necesario estudiar varios casos por separado:

- si $2^N \leq 2a+b<2^{N+1}$, $\boldsymbol{c_1C_0=01}$ y la entrada de la segunda etapa es $2a+b-2^N$, con lo que para ésta:

$$(2a + b - 2^N) + 2^N - m = 2a + b - m < 2^{N+1} - m < 2^{N+1}$$

Cuando $2a+b-m<2^N$, $\boldsymbol{c_2=0}$ y la tercera etapa calcula:

$$(2a + b - m) + 2^N - m = 2a + b + 2^N - 2m \geq 2m + 2^N - 2m = 2^N$$

y necesariamente $\boldsymbol{c_3=1}$; así, la combinación $\boldsymbol{c_1C_0c_2c_3=0101}$ da como resultado $S3$, como ocurre en la tabla 6.4.

Por otra parte, si $2a+b-m \geq 2^N$, $\boldsymbol{c_2=1}$ y la entrada al tercer sumador es $2a+b-m-2^N$; de este modo, para la tercera etapa:

$$(2a + b - m - 2^N) + 2^N - m = 2a + b - 2m < 3m - 2m = m < 2^N$$

con lo que c_3=**0** y $c_1C_0c_2c_3$=**0110** también ha de seleccionar a $S3$ como resultado correcto, como queda reflejado en la tabla 6.4;

- finalmente, el último caso a estudiar corresponde a $2a+b \geq 2^{N+1}$. Sin embargo, es obvio que:

$$2a + b < 3m < 3 \cdot 2^N = 2^{N+2} - 2^N$$

y la única posibilidad es:

$$2^{N+1} \leq 2a + b < 2^{N+2} - 2^N$$

y necesariamente c_1C_0=**10**. Además, la entrada al segundo sumador es $2a+b-2^{N+1}$, con lo que en la segunda etapa se calcula:

$$(2a + b - 2^{N+1}) + 2^N - m = 2a + b - 2^N - m < 3m - 2^N - m =$$
$$= 2m - 2^N < 2^{N+1} - 2^N = 2^N$$

De este modo, c_2=**0** y para el tercer sumador:

$$(2a + b - 2^N - m) + 2^N - m = 2a + b - 2m < 3m - 2m = m < 2^N$$

con lo que c_3=**0**, y $c_1C_0c_2c_3$=**1000** es la última combinación que selecciona $S3$ como resultado correcto, al igual que en la tabla 6.4.

De este modo, se ha probado que las combinaciones de $c_1C_0c_2c_3$ que aparecen en la tabla 6.4 son las únicas posibles y que éstas indican cuál es la salida correcta, q.e.d.

Por tanto, la estructura de la figura 6.9 proporciona de una manera sencilla el sumador requerido para implementar la arquitectura DA-RNS de la figura 6.8. Al igual que para el sumador introducido en el apartado 6.2.2, la estructura de la figura 6.9 puede ser fácilmente segmentada, sin más que incluir los pertinentes registros entre las diferentes etapas. Sin embargo, dado que el sumador va a ser incluido en un bucle, la latencia del cauce de cuatro o cinco ciclos de reloj produciría un efecto muy perjudicial; de este modo,

para una implementación como la de la figura 6.8, la estructura no segmentada es la más conveniente. Por otra parte, para una máxima velocidad [GAR99] del circuito de la figura 6.9 y, dado que de la tabla 6.4 se deduce que la elección entre $S1$ y $S2$ no depende de c_3, el multiplexor de la figura 6.9 puede implementarse con dos multiplexores de dos entradas; uno de ellos seleccionaría entre $S1$ y $S2$ en función de los valores c_1, C_0 y c_2 y podría operar al mismo tiempo que la tercera etapa calcula $S3$; mientras tanto, otro multiplexor de dos entradas seleccionaría entre la salida del anterior y $S3$ en función de c_1, C_0, c_2 y c_3.

6.3.4. Construcción de filtros FIR basados en DA-RNS sobre dispositivos programables

La principal aplicación de la aritmética distribuida es el cálculo de productos internos o sumas de productos y donde más éxito ha encontrado ha sido en el desarrollo de filtros FIR [TAY86, WHI89]. Por otra parte, la aritmética distribuida se ha convertido en los último años en una herramienta útil [GOS94, GOS95] para la implementación de estructuras para DSP, en especial, filtros FIR, sobre dispositivos programables, dado que permite reducir los recursos requeridos por este tipo de aplicaciones y enmascarar las prestaciones limitadas de tales dispositivos en lo que se refiere a la aritmética. Por ello, se ha elegido la implementación de filtros FIR para la demostración de las ventajas en el uso de la estructura DA-RNS propuesta, empleando una vez más la familia FLEX10K de Altera.

Un filtro FIR de K etapas corresponde a la ecuación [TAY98]:

$$y[m] = \sum_{k=1}^{K} H_k x[m-k+1] \qquad \textbf{(6.20)}$$

donde $\{x[m]\}$ e $\{y[m]\}$ son las secuencias de entrada y salida, respectivamente. Por tanto, la ecuación (6.20) corresponde a un cálculo del tipo (6.10), o (6.15) si consideramos su implementación sobre un RNS; de esta manera es posible plantear una implementación basada en DA-RNS.

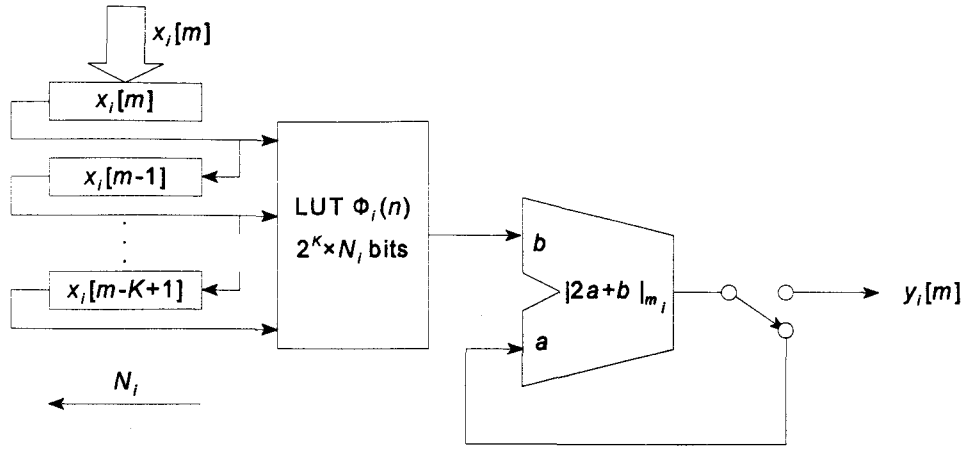


Figura 6.10. Canal módulo m_i de la implementación DA-RNS de un filtro FIR.

La figura 6.10 muestra el canal módulo m_i de la estructura DA-RNS que implementa la ecuación (6.20), donde $x_i[m] = x[m] \bmod m_i$, $y_i[m] = y[m] \bmod m_i$ y $\Phi_i(n)$ está definida por la ecuación (6.18) considerando los coeficientes H_k del filtro. Cada N_i ciclos se carga en paralelo una nueva muestra de la secuencia de entrada en el primer registro de desplazamiento, que irá pasando de uno a otro registro al realizar los desplazamientos necesarios para procesar cada conjunto de datos; se obtiene así, tras N_i ciclos, el resultado $y_i[m]$ correspondiente, con lo que es posible cargar un nuevo dato de entrada.

Para comparar las prestaciones de la arquitectura DA-RNS propuesta en la figura 6.10 con la equivalente basada en aritmética binaria en complemento a dos se van a considerar tres filtros diferentes:

- filtro de 8 bits de entrada, 8 etapas y coeficientes de 13 bits, con lo que el rango dinámico total es de 24 bits;

- filtro de 8 bits de entrada, 8 etapas y coeficientes de 21 bits, con un rango dinámico total de 32 bits;

- filtro de 8 bits de entrada, 10 etapas y coeficientes de 20 bits, con lo que el rango dinámico total será de 32 bits.

Para la implementación RNS de estos tres casos es necesario elegir dos conjuntos de módulos que cubran los rangos dinámicos de 24 y 32 bits. En la sección 6.2 se vio que los módulos de 6 bits son los más adecuados para la implementación de aplicaciones basadas en el RNS sobre dispositivos FLEX10K. Para aquellos filtros con rango dinámico de 24 bits se elegirá el conjunto de módulos {63, 61, 59, 55}, mientras que para el rango de 32 bits la elección será {256, 63, 61, 59, 55}; como ya se comentó en el apartado 6.2.3, la inclusión del canal módulo 256 no limitará las prestaciones del sistema, ya que puede ser implementado como un canal binario convencional de 8 bits.

6.3.5. Resultados experimentales

Con lo dicho en el apartado 6.3.4, la tabla 6.5 muestra los resultados de coste y velocidad para los tres filtros ejemplo propuestos, obtenidos a partir de la síntesis VHDL de los correspondientes sistemas.

TABLA 6.5

		8 etapas, 24 bits		8 etapas, 32 bits		10 etapas, 32 bits	
		DA-RNS	**Binario**	**DA-RNS**	**Binario**	**DA-RNS**	**Binario**
Velocidad (MSPS)	**-4**	3.28	3.27	3.28	2.44	2.78	2.34
	-3	4.22	4.61	4.22	3.79	3.60	3.09
LE		442	170	554	202	648	224
EAB		4 6144 bits	2 4096 bits	5 8192 bits	3 6144 bits	16 32768 b.	12 24576 b.

Los datos de velocidad se han calculado para dispositivos de grado 4 y grado 3, indicado con -4 y -3, respectivamente, en la tabla. Se pueden extraer algunas conclusiones interesantes de la tabla 6.5:

- Las ventajas en velocidad de la arquitectura DA-RNS sobre la alternativa binaria se incrementan conforme crece el orden del filtro, es decir, el número de etapas.

- Cuando el número de etapas es constante y se aumenta el rango dinámico, la velocidad de la arquitectura DA-RNS no sufre penalización alguna, ya que sólo es necesario añadir canales adicionales al sistema; por contra, la alternativa binaria se ve claramente perjudicada cuando se requiere mayor precisión.

- El hecho de considerar dispositivos más rápidos (grado 3 frente a grado 4) resulta en mayores beneficios en velocidad para la alternativa binaria en complemento a dos; esto es así debido a que las cadenas de acarreo de mayor longitud ponen de manifiesto la reducción en el tiempo característico del dispositivo empleado. De este modo, la arquitectura DA-RNS no requiere dispositivos más rápidos, generalmente de coste más elevado, para unas prestaciones dadas.

- Aunque la arquitectura DA-RNS supone un incremento en los recursos necesarios para la implementación de cada uno de los ejemplos propuestos, éstos se reparten entre canales independientes. Esto es especialmente significativo en lo que se refiere al uso de memoria; de este modo, para el filtro de 10 etapas y 32 bits, aunque la arquitectura DA-RNS requiere 16 EABs frente a los 12 EABs de la alternativa binaria, estos últimos forman una sola tabla, mientras que en el caso de DA-RNS corresponden a 5 tablas de menor tamaño. Esto permite usar varios chips de tamaño medio o pequeño para construir filtros de orden elevado sin afectar a las prestaciones, mientras que la alternativa binaria necesitaría un chip de tamaño considerable para mantener las máximas prestaciones.

Por tanto, la arquitectura DA-RNS propuesta en esta sección constituye un método eficiente para la fusión de la aritmética distribuida con el RNS, consiguiendo mejoras de hasta un 35% sobre la implementación binaria basada en aritmética distribuida de filtros FIR sobre dispositivos programables.

UNIVERSIDAD DE GRANADA
13 MAYO 1999
COMISION DE DOCTORADO

6.4. CONCLUSIÓN

En este capítulo se ha demostrado que el RNS es una alternativa válida y eficiente

para la implementación de aplicaciones de procesamiento digital de señales sobre dispositivos programables. En concreto, se ha mostrado que la utilización del RNS sobre la familia FLEX10K de Altera se traduce en ventajas considerables para la implementación de aplicaciones como los filtros CIC de Hogenauer; para ello se han empleado de manera novedosa algunas de las ideas relativas al escalado expuestas en el capítulo V.

Por otra parte, se ha presentado un nuevo esquema para la implementación de aplicaciones basadas en la realización de productos internos uniendo las ventajas de la aritmética distribuida para el empleo de dispositivos programables con las del RNS. De este modo, se ha introducido la arquitectura DA-RNS y se han mostrado las ventajas de la misma frente a la implementación binaria en complemento a dos con aritmética distribuida.

CAPÍTULO VII

NUEVAS ESTRATEGIAS DE SINCRONIZACIÓN VLSI PARA APLICACIONES RNS

Este capítulo muestra una nueva propuesta para la sincronización de circuitos ASIC VLSI basados en el uso del RNS. Esta propuesta aprovecha las características del RNS relativas a modularidad e independencia entre canales a fin de facilitar la sincronización de circuitos VLSI.

7.1. INTRODUCCIÓN

A lo largo de esta memoria se ha presentado el RNS como una alternativa a los circuitos aritméticos convencionales para conseguir mayores prestaciones en la implementación de aplicaciones basadas en la realización de elevados números de sumas y productos; además, esto ocurre para circuitos ASIC [SMI95], dispositivos programables, [GAR98a], etc.

Sin embargo, la creciente capacidad de integración y la mejora de los procesos tecnológicos asociados a la fabricación de circuitos integrados permiten construir sistemas cada vez mayores, más complejos y más rápidos. De este modo, aparecen nuevos problemas al tiempo que la mejora en velocidad a través de arquitecturas especiales deja de ser el factor determinante para algunas aplicaciones; entre los nuevos problemas se

pueden citar los relacionados con la distribución del reloj [AFG90, YUA89] y el mayor consumo de potencia, al ser los circuitos más complejos. La sincronización adecuada de circuitos CMOS VLSI es un problema de complejidad creciente, al aumentar la velocidad de los circuitos, el número de transistores y el área de silicio; así, el retardo asociado a las cada vez más largas líneas de distribución de reloj puede limitar la velocidad de funcionamiento del circuito, mientras que la inclusión de líneas de menor retardo encarece y complica el diseño del sistema.

En este capítulo se va a presentar una nueva estrategia para la sincronización de sistemas basados en el RNS o en cualquier otro esquema aritmético con canales paralelos independientes. Se mostrará que las características del RNS permiten reducir los picos de corriente en las líneas de alimentación de un circuito integrado y suavizar la variación temporal de la misma, al tiempo que permiten una distribución más sencilla de la señal de reloj; estos dos factores permiten escalar las respectivas líneas y simplificar el diseño de circuitos integrados.

7.2. NUEVA ESTRATEGIA DE SINCRONIZACIÓN

El RNS y otros sistemas similares [WIG90] han sido empleados, como se ha indicado a lo largo de esta memoria, para obtener sistemas más rápidos que los basados en aritmética convencional por su habilidad para realizar cálculos sobre un elevado rango dinámico sobre una serie de canales independientes de menor rango dinámico; la independencia entre canales viene dada por el hecho de que no existe propagación de acarreos entre canales, lo que a su vez permite la construcción de sistemas de mayor velocidad. Sin embargo, las aplicaciones clásicas no han hecho un uso exhaustivo de esta propiedad de independencia entre canales; de este modo, es posible que cada canal funcione con un reloj diferente al resto de canales, siendo solamente necesario que todos los canales provean los datos correspondientes al mismo tiempo.

7.2.1. Sincronización de canales independientes

Los sistemas basados en el RNS están formados por una serie de copias del sistema original restringidas a la operación sobre cada uno de los módulos que definen el RNS en cuestión. En lo que se refiere a implementaciones VLSI de estos sistemas, se han empleado diversas alternativas, incluyendo arquitecturas sistólicas [TAH88, SMI95], arquitecturas neuronales [ZHA90], autómatas celulares [PRI86, YOR91], estructuras en árbol [WAN96], lógicas multivaluadas [KAM79, KAM89], etc. Sin embargo, hasta muy recientemente no ha surgido el interés por considerar las implicaciones del RNS en campos como el consumo de potencia de los circuitos integrados [BHA98]. Por otra parte, el otro gran problema en los circuitos integrados de última generación, además de los aspectos relacionados con el consumo, es la distribución de la señal de reloj; la independencia entre canales del RNS puede desempeñar un papel clave en la resolución de algunos de estos problemas y otros relacionados, pero hasta la fecha no existen propuestas para la sincronización independiente de los diferentes canales que componen un sistema basado en el RNS.

De este modo, dado que los canales que forman un sistema basado en el RNS son completamente independientes, se propone aquí un nuevo esquema de sincronización para circuitos VLSI basados en el RNS; esta nueva estrategia de sincronización se basa en asignar un reloj diferente a cada canal del sistema, de modo que sólo es necesario sincronizar las entradas y las salidas de los diferentes canales. En concreto, las condiciones a satisfacer por los diferentes relojes en el circuito son:

- el número de señales de reloj con ciclos activos solapados en todo o en parte ha de ser mínima;
- el tiempo en que dos o más ciclos activos se solapan ha de ser el mínimo posible;
- los flancos de los relojes no han de coincidir;

- la estrategia de sincronización ha de respetar la correspondencia de los diferentes conjuntos de datos; la aplicación de diferentes señales de reloj a los canales del sistema ha de respetar la sincronización de los diferentes conjuntos de datos a la entrada y a la salida del sistema, en especial para sistemas con cauce segmentado.

Con todo esto se obtienen ciertas ventajas:

- no todos los dispositivos en el sistema están gobernados por el mismo reloj; así, no todos funcionan al mismo tiempo, con lo que se reducen los picos de corriente en las líneas de alimentación del circuito; esto implica que la variación temporal de la corriente en las líneas de alimentación del circuito, di/dt, se suaviza;

- como consecuencia, las dimensiones de las líneas de alimentación pueden escalarse, lo que simplifica el diseño del sistema;

- se elimina la distribución de una señal global de reloj por todo el chip; por el contrario, cada canal es gobernado por su propio reloj, con lo que sólo es necesario distribuir cada uno de estos relojes en los recursos del canal correspondiente;

- en consecuencia, se reducen las líneas de conexión y distribución de reloj en el sistema, simplificando la labor de diseño.

Los puntos anteriores definen la nueva estrategia de sincronización, al tiempo que resumen sus ventajas. Si bien las condiciones a satisfacer pueden parecer restrictivas, en especial el combinar señales de reloj diferentes para cada canal con sincronización global de entradas y salidas, se va a mostrar cómo la nueva estrategia de sincronización es factible con unos recursos muy limitados. En concreto, cuando el número de señales de reloj a generar no es demasiado elevado, es posible cumplir estas especificaciones sin más que introducir un ligero retardo entre cada uno de los relojes del sistema; esto se puede conseguir sin más que introducir la señal de reloj global del sistema en una cadena de inversores que alternen altas capacidades de entradas y pobres características de conducción de corriente, y extraer en determinados puntos de la cadena las señales de reloj

mediante buffers adecuados.

En el siguiente apartado se va a describir la estructura básica de la celda para la generación de los diferentes relojes requeridos para la estrategia de sincronización propuesta.

7.2.2. Generación de señales de reloj retardadas

La estrategia de sincronización propuesta puede implementarse mediante la generación de diferentes señales de reloj retardadas con respecto de la señal global de reloj del sistema. Para ello se propone la estructura de la figura 7.1, con la celda parametrizable `dCLK_cell`, asumiendo un proceso CMOS de 0.6μm, como el CMOS14TB (v. apéndice C) que ya se ha usado en esta memoria. Esta celda consta de tres inversores y los parámetros de diseño son L_d, W_d y W_b; la entrada CHin ha de estar conectada a una señal de reloj o a la salida CHout de una celda `dCLK_cell` previa, mientras que la salida dCLK proporciona una señal de reloj retardada en función de los parámetros L_d y W_d, al tiempo que el parámetro W_b define la característica de conducción del buffer de salida. De este modo:

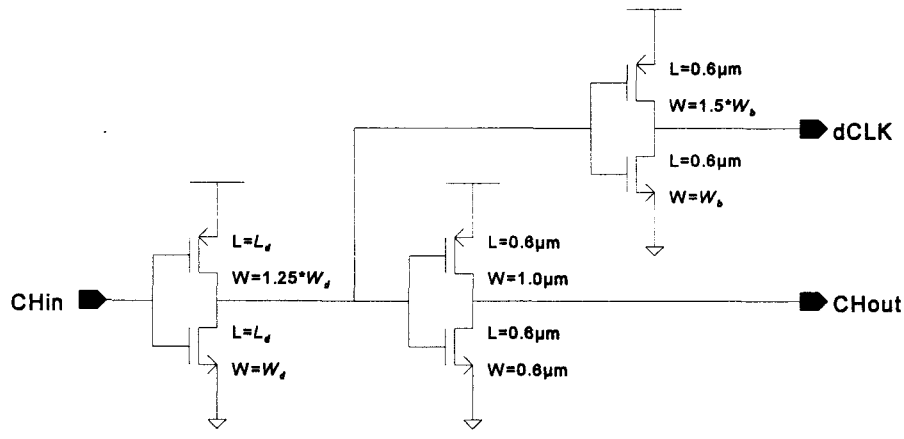


Figura 7.1. Esquemático de la celda `dCLK_cell`.

- El segundo inversor de la cadena es de tamaño mínimo, por lo que proporciona niveles bajos de corriente. Los parámetros L_d y W_d permiten definir la capacidad de entrada del siguiente inversor en la cadena de inversores; por tanto, con L_d y W_d se puede ajustar el retardo producido en la propagación de la señal de reloj a lo largo de la celda `dCLK_cell`.

- Por otra parte, dado que se alternan inversores con gran capacidad de entrada con inversores de tamaño mínimo, las señales generadas en la cadena no son directamente aplicables como señales de reloj: es necesario un buffer para regenerar dCLK a partir de una señal intermedia de la cadena de inversores. La capacidad de entrada de este inversor también influirá en el retardo generado por la celda `dCLK_cell`, aunque en menor medida que L_d y W_d; además, el parámetro W_b permite variar la calidad de la señal de reloj generada en dCLK en función de la capacidad de carga aplicada a este nodo del circuito.

La tabla 7.1 muestra, en función de L_d, el consumo de potencia de la celda `dCLK_cell` y el retardo producido en dCLK con una capacidad de carga de C_L=10fF, respecto a una señal de reloj aplicada a CHin. Estos resultados se han conseguido por simulación eléctrica con spectreS para W_d=4.8μm y W_b=1.6μm. Se puede observar que el retardo producido en la celda crece conforme se aumenta la capacidad de entrada en el primer inversor a través del parámetro L_d, como era de esperar; al mismo tiempo, el consumo también aumenta al hacerlo el retardo, aunque su valor es bastante moderado en todos los casos.

TABLA 7.1

L_d	1.2μm	2.0μm	2.4μm
Retardo en dCLK	0.58ns	0.95ns	1.15ns
Consumo	465μW@125MHz	568μW@125MHz	624μW@125MHz

La figura 7.2 muestra las señales generadas por una cadena de cinco celdas `dCLK_cell` en las mismas condiciones de simulación y para L_d=2.0μm. Se puede observar que a partir de la señal de reloj de entrada CLK se generan cinco señales de reloj retardadas con respecto a ésta y retardadas entre sí, de manera que los flancos de las diferentes señales no coinciden; de este modo, se satisfacen las condiciones impuestas para esta nueva estrategia de sincronización, ya que los flancos de los diferentes relojes no se solapan, al tiempo que se reduce el número de flancos activos solapados; además, dado que los ciclos activos de las señales generadas se producen antes del siguiente ciclo activo de CLK, la sincronización de entradas y salidas globales en un sistema gobernado por estas señales es muy sencilla. Como ejemplo, para un sistema basado en un RNS con cinco módulos y, en consecuencia, cinco canales, funcionando con segmentación de cauce, el reloj global CLK puede controlar la adquisición de los datos de entrada y la carga de los registros de salida, mientras que cada uno de los canales RNS sería gobernado por una de las señales dCLK generadas mediante la estrategia propuesta.

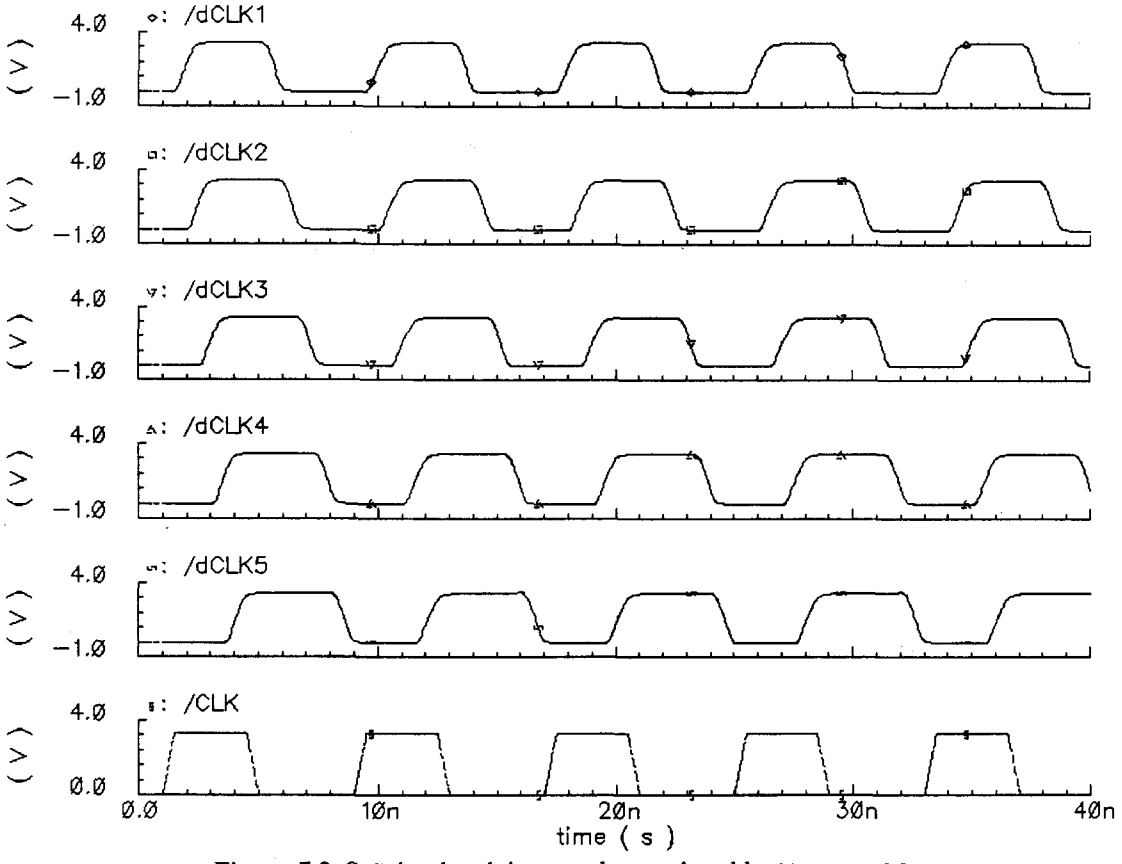


Figura 7.2. Señales de reloj generadas con la celda `dCLK_cell`.

La principal ventaja de la alternativa propuesta es su simplicidad; además, una vez elegidos los parámetros de diseño adecuados para una aplicación en concreto, sólo es necesario el diseño de una celda.

7.2.3. Otras alternativas de generación

En esta sección se han definido las características a satisfacer por la estrategia de sincronización propuesta y se ha mostrado una manera eficiente y sencilla de generación de las diferentes señales de reloj. Sin embargo, aunque la simplicidad de diseño es una de sus principales ventajas, esta propuesta presenta algunas limitaciones: en primer lugar, dado que el retardo asociado a la celda `dCLK_cell` es función del tamaño de los transistores, cuando es necesario controlar grandes capacidades, es decir, sistemas complejos, el retardo puede resultar excesivo; por otra parte, cuando el número de canales es elevado, puede resultar dificultosa la generación de una señal para cada uno de los canales, lo que puede limitar la velocidad de funcionamiento del sistema si se desea mantener un funcionamiento independiente de cada uno de los canales.

Aunque estas limitaciones pueden no ser excesivamente importantes para aplicaciones basadas en el RNS, existen otros sistemas aritméticos similares, como el MRRNS [WIG90, WIG94], en los que el número de canales es más elevado que en el RNS; de este modo, la propuesta presentada en el apartado 7.2.2 puede no ser la más conveniente en estos sistemas para conseguir la estrategia de sincronización descrita en el apartado 7.2.1. Aunque es posible generar una cantidad limitada de señales de reloj y controlar varios canales con una misma señal, esta alternativa reduce las ventajas de la sincronización independiente de canales. Por tanto, a fin de mantener estas ventajas, es posible modificar el esquema de generación de señales de reloj, asignando un buffer independiente para cada canal y usando el reloj global del sistema como entrada de cada uno de estos buffers; así, ligeras diferencias de diseño en estos buffers hacen que los retardos asociados a cada uno de ellos sean diferentes, tal y como se pretende. Sin embargo, la necesidad de realizar un diseño diferente para cada uno de los canales del sistema aumenta la complejidad del sistema y la dificultad de diseño.

7.3. SIMULACIÓN DE LA NUEVA ESTRATEGIA

En esta sección se van a exponer los resultados de simulación para la estrategia de sincronización propuesta en los apartados 7.2.1 y 7.2.2 para un sistema de prueba realista. Se mostrarán las ventajas que se derivan del uso de este nuevo esquema de distribución de reloj en circuitos ASIC VLSI basados en el RNS.

7.3.1. Descripción del sistema

Se ha mostrado a lo largo de esta memoria que en aplicaciones típicas del RNS (v. capítulo VI) son necesarios cuatro o cinco canales; por tanto, se ha usado un sistema de prueba de cinco canales para la estrategia de sincronización propuesta. El sistema de prueba consta de cinco canales, cada uno de ellos formado por tres registros de 8 bits. Para la implementación de estos registros usaremos biestables D disparados por flanco de bajada, o nETDFF (negative Edge-Triggerd D Flip-Flop) [AFG90], basados en un reloj de fase única TSPC [JIR87, YUA89]. La sincronización TSPC es una de las más eficientes para su uso en circuitos CMOS para el desarrollo de sistemas VLSI digitales [AFG90] por su buen comportamiento temporal. La figura 7.3 muestra la estructura básica para el nETDFF; dado que las simulaciones se han llevado a cabo utilizando los parámetros

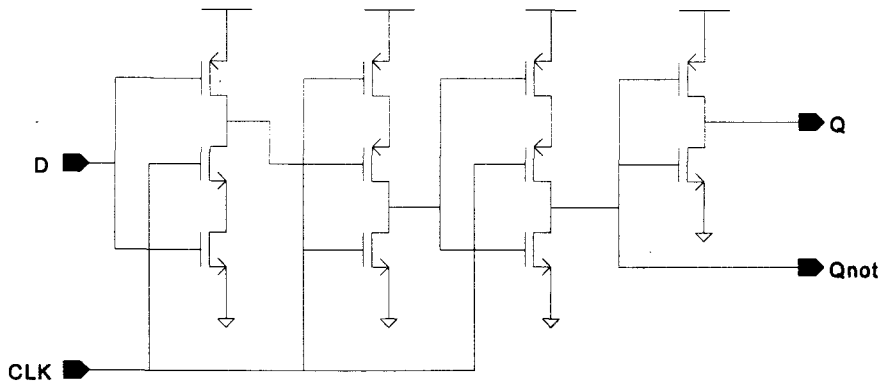


Figura 7.3. Biestable nETDFF basado en TSPC.

propios del proceso CMOS14TB, el tamaño de los transistores NMOS en la figura 7.3 es 1.6μm×0.6μm, mientras que para los PMOS las dimensiones son 2.6μm×0.6μm.

Cada canal está formado por 24 nETDFFs como el de la figura 7.3, con lo que cada canal consta de 264 transistores, 72 de ellos conectados directamente a la señal de reloj del canal. Por otra parte, podemos generar las cinco señales de reloj a partir de una cadena de cinco celdas `dCLK_cell`, de forma que el circuito total de simulación consta de 1350 transistores. Realizando un cuidadoso estudio de los valores idóneos para los parámetros de diseño de la celda `dCLK_cell`, se llega a la conclusión de que éstos son:

$$L_d = 1.0\,\mu m \qquad W_d = 2.0\,\mu m \qquad W_b = 9.0\,\mu m \tag{7.1}$$

Con estos valores de L_d y W_d, el valor de W_b ha de escogerse en función de la calidad de las señales a generar y el retardo deseado. Para el sistema en cuestión, la simulación previa demuestra que W_b=7.0μm resulta en una señal de reloj de baja calidad, con tiempos de subida y bajada excesivos, mientras que W_b=10.0μm produce un retardo excesivo, que por ejemplo, impide aplicar un reloj global de 125 MHz y generar las cinco señales dCLK en el mismo ciclo de reloj. De este modo, W_b=9.0μm es el valor idóneo para el estudio del ejemplo propuesto.

Por otra parte, el mismo sistema de cinco canales de tres registros de 8 bits ha sido simulado con una única señal de reloj, en dos situaciones diferentes:

- Aplicación directa a las líneas de reloj de una fuente de pulsos generando la señal de reloj del sistema; de este modo, aunque la situación no es realista, permitirá comparar los consumos de potencia exclusivamente asociados a la generación de señales de reloj con la estrategia de sincronización propuesta.

- Regeneración de la señal de reloj global del sistema a través de un buffer apropiado; con esta segunda alternativa es posible obtener resultados realistas para el sistema gobernado por un solo reloj en lo que se refiere a los valores de corriente en las líneas de alimentación del sistema.

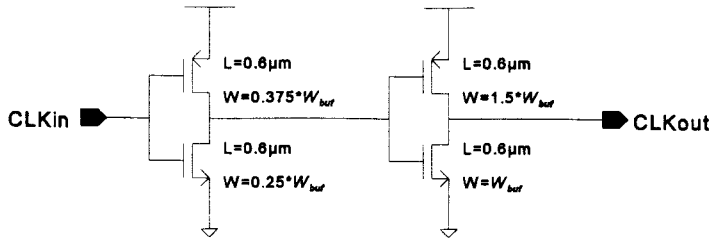


Figura 7.4. Buffer para la regeneración de la señal de reloj.

Para esta segunda alternativa es necesaria la inclusión en el circuito de un buffer adecuado para la regeneración de la señal de reloj a partir de una fuente ideal de pulsos; para ello se empleará un buffer de una sola etapa con dos inversores, de modo que los transistores habrán de poseer las dimensiones adecuadas para la generación de una señal de calidad. El buffer empleado se muestra en la figura 7.4, donde W_{buf} es un parámetro de diseño y habrá de elegirse en función de la capacidad de carga de este buffer y la calidad deseada para la señal generada por el mismo. Para el caso de estudio propuesto, y tras un estudio de las simulaciones previas, se puede elegir el valor de este parámetro de diseño como:

$$W_{buf} = 60.0\,\mu m \tag{7.2}$$

Este valor de W_{buf} es suficiente para generar la corriente necesaria para sincronizar el sistema de cinco canales sin mayores problemas.

7.3.2. Resultados experimentales

Una vez descritos los diferentes circuitos empleados para la simulación de la estrategia de sincronización propuesta y su comparación con sistemas haciendo uso de un solo reloj, este apartado muestra los resultados obtenidos. Hay que recordar que estas simulaciones se han llevado a cabo con spectreS empleando los parámetros del proceso CMOS14TB.

En primer lugar, la figura 7.5 muestra las señales dCLK generadas por la cadena

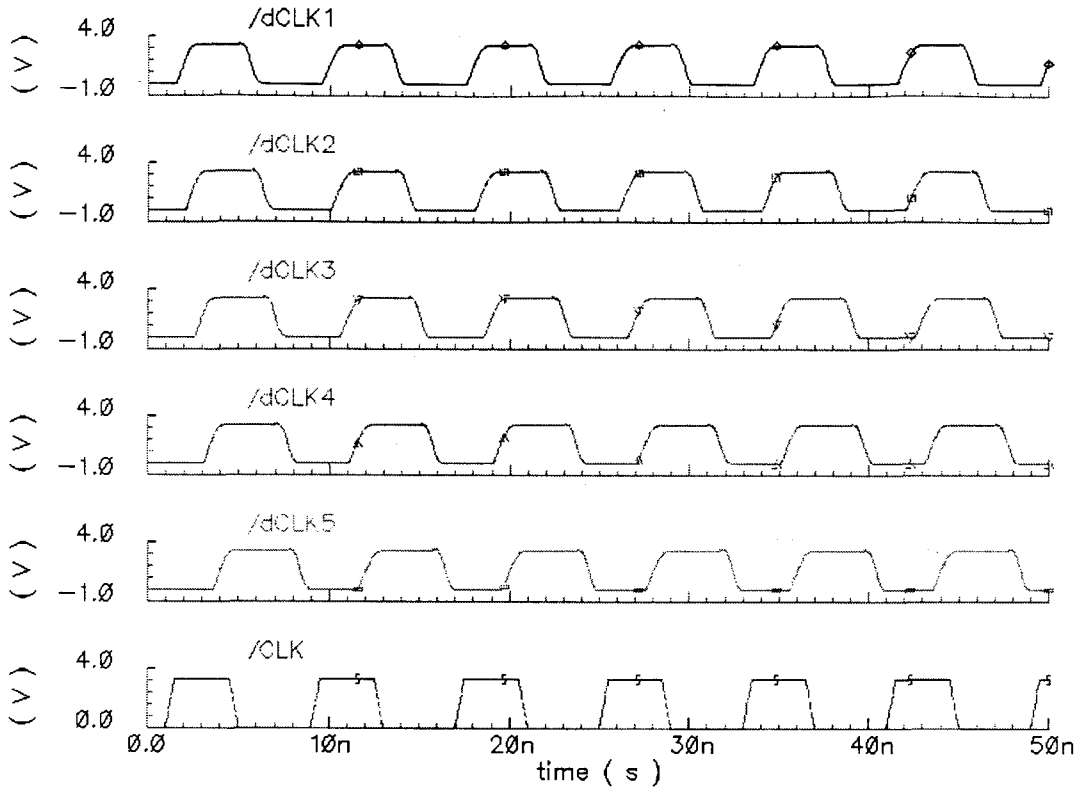
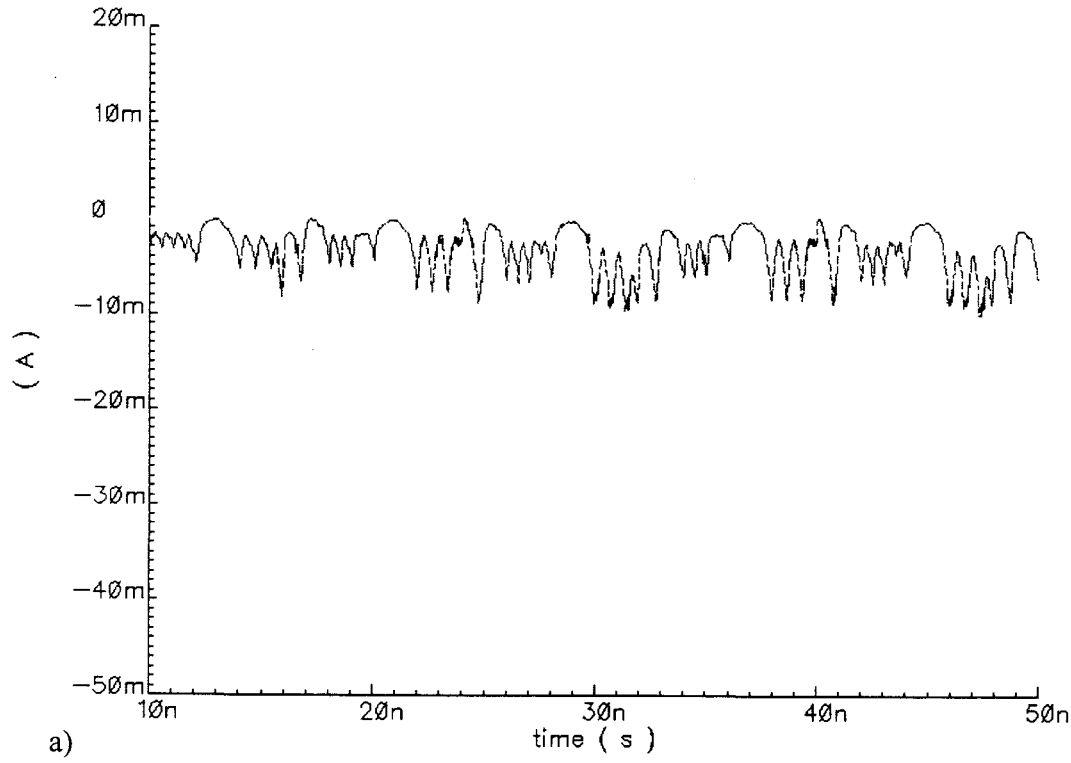


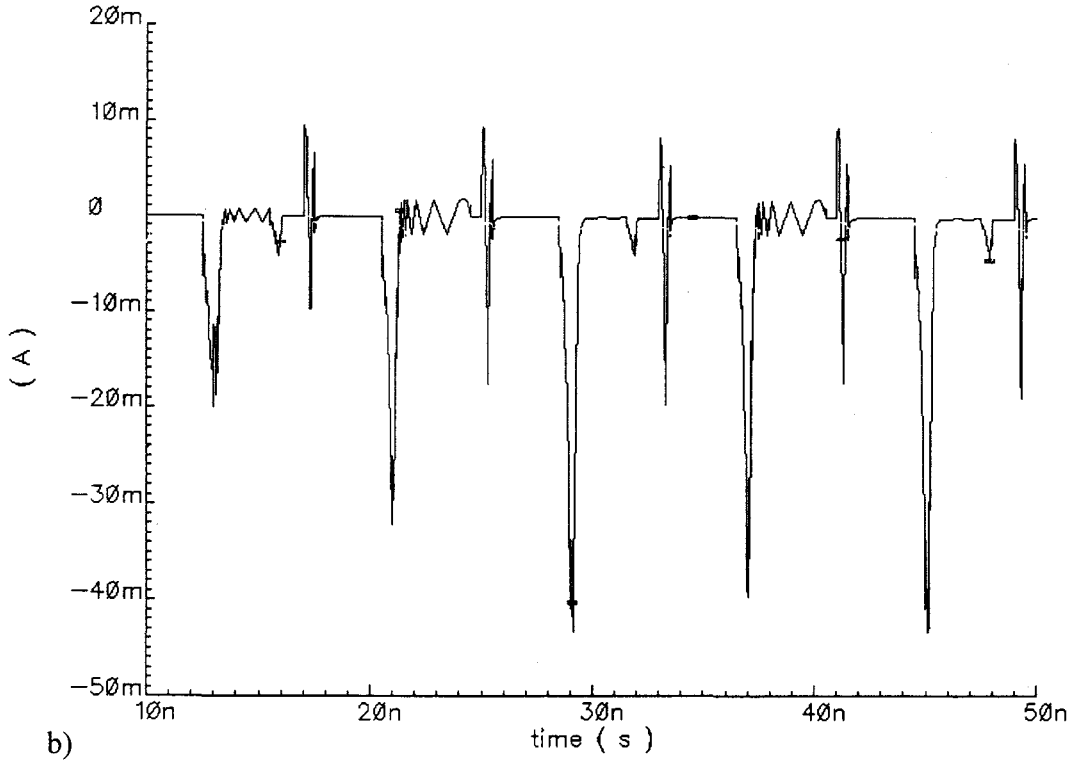
Figura 7.5. Señales dCLK generadas para el sistema de prueba de cinco canales.

de celdas `dCLK_cell` para el sistema de cinco canales descrito en el apartado anterior y con los valores de los parámetros de diseño mostrados en (7.1). Se puede apreciar que las señales de reloj satisfacen las condiciones del esquema de sincronización propuesto en este capítulo, con señales retardadas y flancos no solapados; en concreto, el retardo entre dos señales dCLK consecutivas en la cadena es 0.7ns.

Por otra parte, como se mencionó en la sección 7.2, una de las principales ventajas de la aplicación de esta nueva estrategia de sincronización es la reducción de los valores de corriente que circulan por las líneas de alimentación; de este modo, la figura 7.6 muestra los valores de corriente que proporciona la fuente de alimentación Vdd para el sistema sincronizado según la nueva estrategia y el sistema con un solo reloj regenerado a través de un buffer. Es evidente que la estrategia de sincronización propuesta resulta en una reducción de los valores de corriente que suministra la fuente de alimentación al circuito, lo que se traduce en la posibilidad de emplear líneas de dimensiones más reducidas.



a)



b)

Figura 7.6. Valores de corriente en las líneas Vdd para: a) la nueva estrategia de sincronización; b) sistema con un solo reloj.

La figura 7.6 presenta variaciones más suaves para la nueva propuesta que para el caso de un solo reloj, lo que puede apreciase más detalladamente en la figura 7.7, donde se muestra la variación temporal de la corriente suministrada por la fuente Vdd, d*i*/dt, para ambos casos. De nuevo se aprecia una reducción considerable en este valor para la estrategia de sincronización propuesta frente a la alternativa de un solo reloj.

Finalmente, la tabla 7.2 muestra los valores más destacados de las diferentes simulaciones llevadas a cabo, todas ellas con un reloj global de 125 MHz, y las gráficas

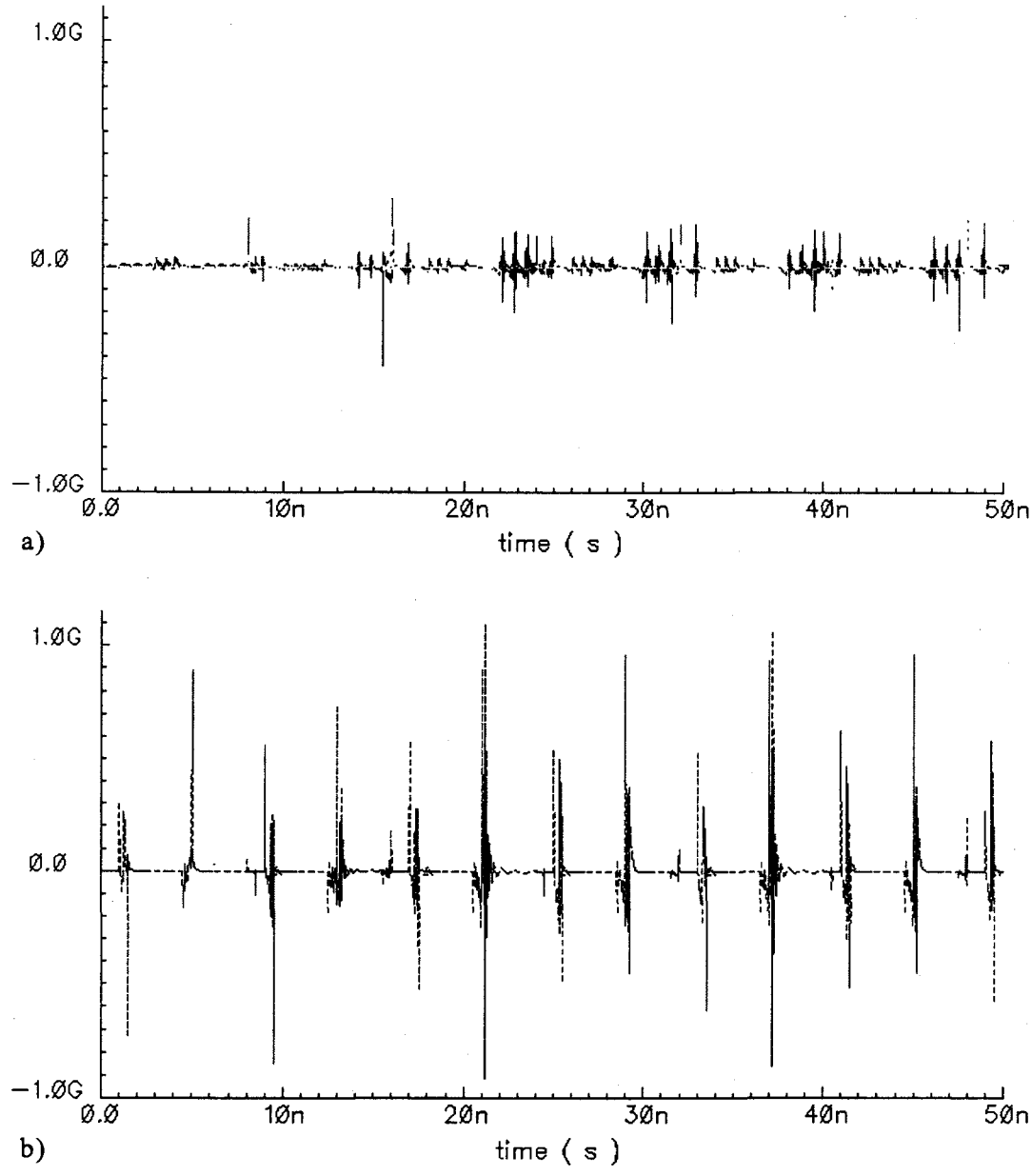


Figura 7.7. d*i*/dt en las líneas Vdd para: a) la nueva estrategia de sincronización; b) sistema con un solo reloj.

mostradas en este apartado; todos los datos de esta tabla están referidos a la fuente de alimentación del circuito Vdd.

TABLA 7.2

	Un solo reloj	**Un solo reloj (con buffer)**	**Relojes generados por `dCLK_cell`**
Máximo pico de corriente	43.326 mA	44.202 mA	9.656 mA
Valor máximo d*i*/dt	1.094 A/ns	942 mA/ns	437 mA/ns
Consumo @125 MHz	8.125mW	10.942mW	10.173mW

De los datos de la tabla 7.2 se puede deducir que:

- Con la nueva estrategia de sincronización, el valor máximo de la corriente proporcionada por la alimentación del circuito se reduce en un factor que es aproximadamente equivalente al número de relojes generados por las celdas `dCLK_cell`.

- De manera similar, y para los parámetros de diseño elegidos en cada caso, la variación temporal d*i*/dt de esta corriente también se ve atenuada, aproximadamente en un factor 2.

- En lo que se refiere al consumo, la comparación con la simulación de un solo reloj tomado de una fuente ideal muestra que la nueva alternativa no supone una penalización frente a la alternativa más realista de regeneración del reloj global con un buffer; además, se confirma que son necesarios entre 400 y 500 μW por cada celda `dCLK_cell` funcionando a 125 MHz.

Por tanto, el uso de la celda `dCLK_cell`, propuesta en el apartado 7.2.2, satisface

las condiciones impuestas para la estrategia de sincronización descrita en el apartado 7.2.1, al tiempo que consigue los objetivos fijados para ésta. De este modo, la reducción en los valores de corriente que la alimentación del circuito ha de proporcionar se traducirá en una simplificación del diseño. Mientras tanto, el empleo de diferentes relojes para cada uno de los canales del sistema elimina la necesidad de distribución de una señal global de reloj por todo el sistema; así, sólo es necesario llevar el reloj del sistema a las celdas encargadas de la generación de los diferentes relojes y a las entradas y salidas de cada uno de los canales, mientras que la distribución de los relojes generados se reduce al área ocupada por el canal que controla cada uno de ellos.

7.4. CONCLUSIÓN

En este capítulo se ha presentado una propuesta para la sincronización de sistemas VLSI basados en el RNS u otros sistemas similares que impliquen cierto número de canales independientes. La estrategia de sincronización propuesta exige la generación de señales de reloj para cada uno de los canales a partir del reloj global del sistema; estas señales han de tener flancos no coincidentes, al tiempo que ha de minimizarse el número de ciclos activos solapados y el tiempo en que lo están. De este modo, es posible reducir los valores máximos de corriente en las líneas de alimentación del circuito y eliminar la necesidad de la distribución de una señal global de reloj por todo el sistema; estos dos factores simplifican el diseño del sistema al ser posible un escalado de las líneas de alimentación y una reducción de las interconexiones del sistema.

Por otra parte, se ha propuesto un esquema de generación de señales de reloj para la estrategia descrita; este esquema es sencillo y consta de una única celda con tres parámetros de diseño que habrán de ser elegidos en función de las características del sistema. Esta propuesta se ha simulado para un sistema de cinco canales usando elementos TSPC y los resultados han sido comparados con los obtenidos para el mismo sistema gobernado por un reloj global; de la comparación se puede concluir que:

- los picos de corriente en la alimentación del sistema se reducen en un factor aproximadamente igual al número de canales considerados;

- la variación temporal d*i*/dt de esta corriente también se ve atenuada;

- todo esto requiere un consumo moderado y no supone penalización alguna respecto de un caso realista con un solo reloj global.

CAPÍTULO VIII

CONCLUSIONES Y PRINCIPALES APORTACIONES

En este último capítulo se enuncian las conclusiones finales de la memoria, indicando las principales aportaciones efectuadas y planteando futuras líneas de investigación.

8.1. INTRODUCCIÓN

El Sistema Numérico de Residuos, más conocido como RNS, se ha convertido en los últimos años [IBR94, CHR95, SMI95] en una alternativa a los sistemas aritméticos convencionales para la implementación de aplicaciones basadas en la realización de un elevado número de sumas y productos. Los motivos de este interés [TAY84] son las especiales propiedades del RNS [SZA67], que le permiten realizar la suma y la multiplicación sobre una serie de canales independientes entre los que no existe propagación de acarreos; de este modo, un sistema trabajando sobre un determinado rango dinámico puede implementarse como un conjunto de copias del sistema original con rangos dinámicos muy reducidos y trabajando en paralelo. El hecho de que el RNS presente dificultades para el desarrollo de sistemas de propósito general no ha limitado su aplicación, ya que la mayoría de algoritmos para procesamiento digital de señales, un campo en constante expansión, están basados en la realización masiva de sumas y

productos.

El objetivo de la investigación recogida en esta memoria ha sido desarrollar nuevas arquitecturas para la realización de las operaciones básicas en el RNS, además de desarrollar nuevas aplicaciones basadas en el RNS. La colaboración con otros grupos de investigación ha abierto nuevas líneas de trabajo durante el período de realización de esta investigación que también se recogen en esta memoria.

8.2. CONCLUSIONES PREVIAS

La revisión exhaustiva de la bibliografía disponible he permitido llegar a una serie de interesantes conclusiones que han servido de base al trabajo realizado.

1. Aunque el RNS ha padecido hasta fechas recientes los inconvenientes derivados de sus requerimientos especiales, lo que ha limitado su aplicación, el avance tecnológico actual solventa gran parte de estas dificultades. De este modo, circuitos ASIC VLSI permiten la integración de las estructuras necesarias para la implementación de aplicaciones basadas en el RNS, mientras que las modernas familias de CPLDs han abierto todo un nuevo campo de aplicación al permitir incluir en un solo chip aplicaciones completas de procesamiento digital de señales; además, el uso de dispositivos programables permite realizar una comparación directa entre las prestaciones de una arquitectura basada en el RNS y su equivalente binaria. Por tanto, el estado actual de desarrollo tecnológico convierte al RNS en una opción en clara superioridad sobre arquitecturas clásicas para la optimización de un gran número de aplicaciones y el desarrollo de nuevas estructuras de procesamiento digital de altas prestaciones.

2. El desarrollo de arquitecturas de alta velocidad no es la única posibilidad de aplicación del RNS; sus características de modularidad e independencia entre canales abren nuevas posibilidades. Aunque estas características no pueden aprovecharse más allá de la mejora en prestaciones cuando se emplean dispositivos

programables, la integración VLSI de sistemas complejos de alta velocidad plantea problemas que el RNS puede contribuir a solucionar.

3. La realización rápida y eficiente de la suma y la multiplicación sobre el anillo definido por el producto de los módulos presenta como contrapartida la inexistencia del desbordamiento; de este modo, el escalado se convierte en una operación crucial a la hora de implementar ciertas aplicaciones de procesamiento digital, a menos que se elija un rango dinámico suficiente, lo que puede resultar en un coste excesivo. Por tanto, nuevos algoritmos eficientes de escalado pueden simplificar en gran medida la traslación de un gran número de aplicaciones al dominio del RNS.

4. La revisión de las arquitecturas existentes para la realización de las operaciones básicas en el RNS muestra que la elección de la alternativa más adecuada depende en cada caso de la tecnología en la que se implementarán estos sistemas. Por ello, se han mostrado en esta memoria algunas de las opciones más representativas entre las existentes en la literatura para ilustrar el amplio abanico de posibilidades disponibles.

5. Finalmente, dada la base matemática del RNS en la teoría de anillos y cuerpos, la aplicación de las propiedades matemáticas de estas estructuras puede ser en muchos casos una herramienta muy útil para la simplificación de las arquitecturas basadas en el RNS.

8.3 PRINCIPALES APORTACIONES

La investigación realizada se ha traducido en una serie de interesantes aportaciones que se han expuesto en esta memoria; dichas aportaciones pueden resumirse como sigue:

1. Se ha propuesto una modificación del multiplicador de mapeo isomórfico submodular; esta modificación permite dotar a dicho multiplicador de

segmentación de cauce al incluir la multiplicación por cero de manera eficiente aprovechando las posiciones no asignadas en las tablas de índices. Esto es posible ya que, en contra de lo afirmado previamente en la literatura, sólo es necesaria la representación del índice de cero para uno de los submódulos del multiplicador.

2. De manera similar, se ha incorporado al multiplicador de mapeo isomórfico submodular un esquema para la detección de errores; la implementación de este esquema de detección de errores sólo requiere que uno de los submódulos satisfaga $m_i \leq 2^{k_i} - 2$. El coste de este esquema de detección de errores y la inclusión de la multiplicación por cero es nulo en lo que se refiere al cálculo de índices y la regeneración del resultado a partir de su índice; pero, lo que es más, si los sumadores para los submódulos están basados en tablas de consulta, el coste global de detección de errores y multiplicación por cero es nulo.

3. De este modo, el multiplicador de mapeo isomórfico submodular con cauce segmentado propuesto en esta memoria puede ser fácilmente implementado sobre dispositivos programables o en un circuito ASIC VLSI.

4. Se ha propuesto un nuevo esquema de escalado basado en el uso de módulos aritméticos convencionales para el RNS; así, se ha demostrado que cuando la constante de escalado es el mínimo de los módulos, esta operación se puede realizar a través de una resta y una multiplicación. Haciendo uso del multiplicador de mapeo isomórfico submodular con cauce segmentado propuesto en esta memoria y del restador de rotación-selección, se ha mostrado que el esquema de escalado propuesto es más ventajoso que las alternativas previas en términos de la relación entre velocidad y coste.

5. Se ha propuesto un nuevo algoritmo de escalado basado en el uso de tablas de consulta. Mientras que los algoritmos existentes en la literatura derivan en realizaciones cuya latencia y uso de memoria crece al aumentar el número de módulos, el algoritmo propuesto posee un tiempo de cálculo fijo de dos ciclos de acceso a memoria; esto es posible ya que se ha demostrado que cuando la

constante de escalado es el producto de varios módulos, el escalado se descompone en dos operaciones independientes que pueden realizarse mediante su almacenamiento en tablas de consulta en ROM; de esta manera, el tiempo de escalado queda fijado en dos ciclos de acceso a reloj. Este algoritmo es aplicable sobre dispositivos programables o circuitos VLSI para conjuntos de módulos pequeños.

6. Se ha mostrado una implementación VLSI del algoritmo de escalado basado en tablas de consulta para un RNS definido por tres módulos de cinco bits con uno de ellos como constante de escalado. Este esquema de escalado requiere tres tablas de 1024×5 bits, para lo que se ha presentado una estructura compacta y de consumo reducido para dichas tablas basada en un proceso CMOS de 0.6μm. El circuito final para el escalado ocupa 0.509 mm^2 y consume menos de 13mW cuando funciona a 125 MHz.

7. Se ha demostrado que es posible combinar el algoritmo de escalado basado en tablas de consulta propuesto en esta memoria con alguno de los algoritmos iterativos propuestos por otros autores; en concreto, se han propuesto dos nuevos algoritmos mixtos. Cuando el algoritmo basado en tablas de consulta supone un uso excesivo de memoria, se ha mostrado que para la mayoría de conjuntos de módulos propuestos alguna de las alternativas mixtas es una solución intermedia entre el algoritmo basado en tablas y las propuestas previas, tanto en uso de memoria como en número de ciclos de acceso a memoria.

8. Se ha desarrollado un sumador en módulo de alta velocidad para su uso sobre dispositivos programables; además, se ha demostrado que para la familia de CPLDs FLEX10K de Altera, esta estructura alcanza sus máximas prestaciones para módulos de seis bits.

9. Haciendo uso del sumador para dispositivos programables propuesto, se ha presentado una implementación basada en el RNS de un filtro CIC para supresión de tres etapas. Se ha demostrado que el uso del RNS para esta estructura se traduce

en una mejora en velocidad superior al 54% con respecto de la implementación binaria del mismo sistema.

10. Se ha presentado una implementación basada en el RNS con reducción del rango dinámico del filtro CIC para supresión de tres etapas; para ello se ha introducido el esquema denominado BRS, consistente en una operación de escalado sin extensión de base, y se ha combinado con el algoritmo ε-CRT. El empleo de esta alternativa supone un ahorro de recursos del 36% frente a la implementación basada en RNS sin reducción de rango dinámico, mientras que se mantiene una ventaja en velocidad del 43% sobre la implementación binaria del sistema.

11. Se ha presentado la arquitectura DA-RNS, consistente en la fusión de la aritmética distribuida con el RNS; para ello, se ha presentado un sumador en módulo que permite adaptar el algoritmo clásico de aritmética distribuida al RNS, evitando los inconvenientes derivados de la operación sobre anillos de números enteros. Además, el algoritmo propuesto inicia el cálculo con los bits más significativos de las entradas, lo que permite trabajar con valores no escalados.

12. Se ha demostrado que la arquitectura DA-RNS propuesta es una alternativa válida para la implementación de aplicaciones basadas en el RNS sobre dispositivos programables. En concreto, esta arquitectura se ha empleado para la implementación de filtros FIR con diferentes números de etapas y precisiones y se ha comparado con las implementaciones binarias basadas en aritmética distribuida, obteniéndose mejoras de hasta el 35% en las prestaciones.

13. Se ha propuesto una nueva estrategia de sincronización para circuitos ASIC VLSI basados en el RNS. La estrategia propuesta permite reducir los picos de corriente y la variación temporal de la misma en las líneas de alimentación del circuito, al tiempo que elimina la necesidad de distribución de una señal de reloj global por todo el chip. De este modo, es posible escalar las dimensiones de las líneas de alimentación del circuito y reducir el número de líneas de distribución de reloj, simplificando el diseño de sistemas VLSI.

14. Se ha demostrado que la estrategia de sincronización propuesta puede ser implementada con un mínimo de recursos, y a través de la simulación de un sistema de cinco canales, se han mostrado resultados concretos que ilustran las ventajas de la nueva estrategia.

8.4 LÍNEAS DE INVESTIGACIÓN FUTURAS

Tras la investigación mostrada en la presente memoria, quedan líneas de investigación abiertas de gran interés:

1. El escalado es una operación crítica en el RNS y la decisión sobre su empleo o no en una cierta aplicación dependerá en gran medida de la tecnología sobre la que se implemente dicha aplicación; de este modo, aunque se han presentado algoritmos de escalado más ventajosos que las alternativas existentes en la literatura, la búsqueda de nuevos algoritmos para el escalado en función de tecnologías concretas puede producir resultados muy interesantes para la optimización de aplicaciones basadas en el RNS.

2. Se ha demostrado que las modernas familias de CPLDs permiten implementar aplicaciones de procesamiento digital de señales empleando el RNS con claras ventajas sobre las alternativas binarias. Por tanto, es posible profundizar en el desarrollo de aplicaciones basadas en el RNS sobre dispositivos programables; dado que en esta memoria se ha prestado especial atención al filtrado digital y la implementación de estructuras FIR, el campo de las transformadas discretas ofrece un jugoso campo de estudio y desarrollo.

3. Por último, la memoria concluye con la exposición de nuevas propuestas para la integración VLSI de sistemas RNS. Así, cabe plantear la profundización en este campo, con el desarrollo de nuevas estructuras orientadas al bajo consumo y la reducción de los problemas asociados a la sincronización de sistemas digitales y la distribución de la señal de reloj y su empleo en aplicaciones reales.

8.5 CONCLUSIÓN

Con la presentación de las principales aportaciones y el esbozo de las líneas de investigación que aún permanecen abiertas finaliza la exposición de la investigación desarrollada por el doctorando y que ha sido presentada en esta memoria. El objetivo con el que se inició esta investigación fue el tratar de aprovechar las ventajas que el RNS ofrece hoy en día para el desarrollo de aplicaciones de procesamiento digital con gran carga computacional; de este modo, se han presentado de manera estructurada las principales características del RNS, tras lo que se han expuesto los resultados del trabajo desarrollado, que se resumen en este capítulo final.

APÉNDICE A

INTRODUCCIÓN AL ÁLGEBRA DE LOS CUERPOS FINITOS

Este apéndice recoge los fundamentos sobre el álgebra de los cuerpos finitos [LIP81], o cuerpos de Galois, cuyas propiedades son empleadas en diversos apartados de esta memoria.

A.1. GRUPOS

A continuación se presenta la definición de las entidades más sencillas que se pueden encontrar:

Definición A.1: *un* **grupoide** $[G; \cdot]$ *es un sistema algebraico con una operación binaria* $\cdot : (x, y) \rightarrow x \cdot y$ *(también notado* xy*), sin ninguna propiedad específica.*

Definición A.2: *un* **semigrupo** $[S; \cdot]$ *es un grupoide que satisface la propiedad asociativa* $x(yz)=(xy)z$.

En la aritmética ordinaria, $[\mathbb{R}; +]$ es un ejemplo de semigrupo; otros semigrupos más interesantes para el campo de aplicación de esta memoria son $[Z_m; \oplus]$ y $[Z_m; \odot]$, con

$a \oplus b=(a+b) \bmod m$ y $a \odot b=ab \bmod m$, y donde Z_m representa los enteros menores que m, $Z_m=\{0, 1, 2,..., m-1\}$.

Definición A.3: *un* **monoide** [M; · , 1] *es un sistema algebraico con una operación binaria* · *y un elemento identidad* 1 *tales que* [M; ·] *es un semigrupo y se satisface la ley de identidad* x1=1x=x.

Ejemplos de monoides son $[Z_m; \oplus, 0]$ y $[Z_m; \odot, 1]$.

A partir de estas definiciones se puede llegar al concepto de grupo, el más importante y estudiado de todos los sistemas algebraicos:

Definición A.4: *un* **grupo** [G; · , $^{-1}$, 1] *es un sistema algebraico con una operación binaria* · , *una operación de un operador* $^{-1}$ *(inverso) y un elemento identidad* 1 *tales que* [G; · , 1] *es un monoide y se satisface la ley inversa* $x^{-1}x=xx^{-1}=1$.

Un ejemplo de grupo es [Z_m ; $\oplus$, $\ominus$, 0], donde $\ominus a=m-a$ si $a \neq 0$ (0 si $a=0$), que en los esquemas aritméticos aditivos usuales equivale a la substracción.

En este punto es preciso señalar que el monoide multiplicativo $[Z_m; \odot, 1]$ no es un grupo (y en general cualquier monoide multiplicativo), ya que no existe un inverso para cero (la operación inversa en este caso corresponde a la división en los esquemas aritméticos usuales). Por tanto, resulta tentadora la posibilidad de eliminar el cero y afirmar que $Z_m^*=Z_m - \{0\}$ es un monoide; desgraciadamente esto no es posible, ya que con $Z_4^*=\{1, 2, 3\}$ tenemos que $2 \odot 2=0$, con lo que la multiplicación deja de ser una operación cerrada. Sin embargo, existen algunos valores de m para los que es posible esta pequeña treta:

Teorema A.2: *Considerando la multiplicación módulo* m,

Z_m^* *es un grupo multiplicativo* $\Leftrightarrow$ m *es* primo.

Demostración: (⇒) si m no es primo, supóngase que $m=st$, con $1<s$ y $t<m$, con lo que $s\odot t=st \bmod m=m \bmod m=0$, con lo que $\odot$ no es cerrada para Z_m^* y éste ni siquiera es un grupoide con $\odot$.

(⇐) sea $m=p$ primo; para demostrar la implicación es preciso probar que Z_p^* es cerrado con $\odot$; para ello supóngase que existen $a,b \in Z_p^*$ y que $ab=0$; de este modo:

$$a \odot b = 0 \quad \Rightarrow \quad (ab) \bmod p = 0$$

que es la condición necesaria y suficiente para que ab sea múltiplo de p; para que esto ocurriera, a ó b debería ser múltiplo de p, lo que es imposible dado que $1 \le a,b<p$. Por tanto, $a\odot b \neq 0$, lo que prueba el cierre.

Por otra parte, también es necesario probar que $\odot$ es asociativa; esto se deduce *a fortiori* del hecho de que $\odot$ es asociativa en Z_p. Por tanto, $[Z_p^*; \odot]$ es un semigrupo. Además, dado que $1 \in Z_p^*$ es el elemento identidad de $\odot$, $[Z_p^*; \odot, 1]$ es un monoide.

Para llegar a la implicación que se quiere probar faltan los inversos multiplicativos; sea $a \in Z_p^*$, y dado que p es primo y $1 \le a<p$ se deduce que a y p tienen sólo a 1 como divisor común, por lo que se puede demostrar que existen dos enteros s y t [LIP81] que satisfacen:

$$1 = \mathrm{mcd}(a, p) = sa + tp$$

Se demuestra que $s \bmod p$ es el inverso multiplicativo de a, dado que:

$$1 = 1 \bmod p = (sa + tp) \bmod p = sa \bmod p =$$
$$= \left[(s \bmod p)(a \bmod p)\right] \bmod p = (s \bmod p) \odot (a \bmod p) = (s \bmod p) \odot a$$

que es precisamente la definición del inverso multiplicativo de a, ya que $(s \bmod p) \in Z_p^*$. Por tanto, $[Z_p^*; \odot, ^{-1}, 1]$ es un grupo, q.e.d.

A pesar del interés que presentan los grupos, se debe seguir avanzando hacia entidades de mayor complejidad.

A.2. ANILLOS Y CUERPOS

El siguiente peldaño en las estructuras algebraicas lo constituye el concepto de anillo:

Definición A.5: *un* **anillo** $[R;+, -, 0, \cdot, 1]$ *es un sistema algebraico que satisface:*

1. $[R;+, -, 0]$ *es un grupo abeliano o conmutativo (el grupo aditivo de* R*);*
2. $[R; \cdot, 1]$ *es un monoide (el monoide multiplicativo de* R*);*
3. $a(b+c)=ab+ac$ *y* $(b+c)a=ba+ca$ *(propiedad distributiva a derecha e izquierda).*

Un anillo se dice conmutativo si su multiplicación es conmutativa ($ab=ba$). Además, existen elementos del anillo que presentan un especial interés:

Definición A.6: *un elemento* $a \neq 0$ *en* R *se dice* **divisor de cero** *si* $ab=0$ *para algún* $b \neq 0$ *en* R *(esto implica que* b *es otro divisor de cero). Un elemento* $u \neq 0$ *en* R *se dice que es una* **unidad** *si* u *es invertible, es decir, existe* $v \in R$ *tal que* $uv=1$ $(v=u^{-1})$.

En principio puede parecer extraño el definir divisores de cero al restringirse a esquemas aritméticos convencionales, pero si se considera, por ejemplo, Z_8, 2 es un divisor de cero, ya que $2 \odot 4=8 \bmod 8=0$; además, 3 es una unidad, ya que $3 \odot 3=9 \bmod 8=1$. Habitualmente, el conjunto de todas las unidades de un anillo R se denota como $U(R)$. Se puede demostrar que en un anillo conmutativo una unidad es también un divisor de cero.

Estos dos conceptos que se acaban de definir permiten introducir dos importantes clases de anillos conmutativos:

Definición A.7: *un* **dominio integral** *es un anillo conmutativo no trivial en el que no existe ningún divisor de cero.*

Definición A.8: *un* **cuerpo** *es un anillo conmutativo no trivial en el que todo elemento no nulo es una unidad. Se deduce rápidamente que un cuerpo es dominio integral.*

El ejemplo tipo de dominio integral es el anillo de los enteros Z, que además no es un cuerpo, ya que 1 y -1 son las únicas unidades. Respecto a los cuerpos, los ejemplos que más habituales son los anillos de números **Q,** $\mathbb{R}$ y $\mathbb{C}$, en los que está perfectamente definida la división y, por tanto, para cada elemento no nulo existe un inverso; sin embargo, se van a ver ahora mismo unos cuerpos mucho más interesantes para lo que aquí se presenta, ya que para todo *p* primo es fácil demostrar [LIP81] que cada elemento de Z_p^* es una unidad; además, si $m=ab$, $ab=0$ en Z_m. Por tanto, se puede enunciar el siguiente teorema, de demostración trivial:

Teorema A.3: *Si* p *es primo,* Z_p *es un cuerpo; en cambio, si* m *no es primo,* Z_m *es un anillo con divisores de cero.*

Los cuerpos representan un papel importante por razones obvias desde un punto de vista operacional, ya que en uno de ellos se pueden llevar a cabo las cuatro operaciones racionales, capacidad que es la base de muchos algoritmos numéricos (reales y complejos), como la solución de sistemas lineales de ecuaciones por eliminación gaussiana [ENC78]. Sin embargo, la importancia de los dominios integrales es menos obvia, aunque bastantes afirmaciones que resultan familiares sobre los números reales y complejos dependen de manera crucial de la propiedad de los dominios integrales de ausencia de divisores de cero.

El cuerpo más familiar para todos es **Q**, el cuerpo de los números racionales, formado por todos los racionales (cociente de enteros) positivos y negativos y el cero, y

con las operaciones de suma, resta, multiplicación y división definidas por las reglas de operación con fracciones. Dado que $m/n=(km)/(kn)$ para cualquier entero positivo k, un mismo número racional puede representarse de muchas maneras como cociente de enteros, aunque sólo una puede considerarse mínima, en la que numerador y denominador son primos entre sí.

El cuerpo de los reales, $\mathbb{R}$, es mucho más complejo que $\mathbf{Q}$ y los números racionales pueden considerarse como aquellos reales con representación decimal periódica. Por tanto, $\mathbf{Q}$ es un subcuerpo de $\mathbb{R}$. Dada la complejidad de la representación de los números reales (se pueden considerar éstos de una manera muy simplista como números con infinitas cifras decimales), las operaciones entre ellos se acercan bastante a la idea de aproximaciones sucesivas. Una de las principales propiedades de $\mathbb{R}$ se refiere a su representación geométrica, entendiendo los números reales como las coordenadas de los puntos que forman una recta indefinida; este tipo de representación facilita la transición de las fracciones a los números reales a través del mecanismo de aproximaciones sucesivas que hemos comentado anteriormente.

En lo que se refiere a $\mathbb{C}$, el cuerpo de los números complejos, es difícil pensar en que se hubiera estudiado alguna vez de no ser por el interés por ecuaciones del tipo $z^2+1=0$, es decir, ecuaciones sin solución real. De hecho, la unidad i se introduce como la solución de esta ecuación y las reglas para llevar a cabo las cuatro operaciones básicas se definen para hacerlas compatibles con las operaciones con números reales. De nuevo, la interpretación geométrica de $\mathbb{C}$ como un plano representa un papel muy importante en el estudio de este cuerpo, dando un significado geométrico a muchas de las operaciones que se realizan en él.

Es posible generalizar la construcción de los cuerpos Z_p y los anillos conmutativos Z_m (teorema A.3); la idea clave de esta generalización se deriva de los siguientes conceptos [ENC78]:

Definición A.9: *dado un anillo conmutativo* R, *un* **ideal** H *es un subconjunto no vacío tal que:*

1. h, h'∈H ⇒ h±h'∈H;
2. h∈H, r∈R ⇒ rh=hr∈H.

Definición A.10: *una* **clase residual** *de un ideal* H *es un conjunto* H+x *formado por todas las sumas* h+x, h∈H, *para un* x∈R *dado.*

Los ideales propios de Z (los triviales son el propio Z y {0}) se notan como (n), y consisten en todos los múltiplos kn, $k\in Z$, para un entero dado $n>1$. Las clases residuales del ideal (n) son $r+(n)$ y están formadas por los enteros x que satisfacen que $x=qn+r$, es decir, r es el residuo en módulo n de x.

Z_n puede considerarse como el conjunto de todas las clases residuales, con reglas para la suma y multiplicación definidas por la aritmética de residuos; cuando se construye de esta manera, Z_n recibe el nombre de ***anillo cociente*** $Z/(n)$ de Z sobre el ideal (n).

Estos últimos conceptos permiten dar un paso adelante:

Definición A.11: *si* F *es un cuerpo, y* p(x) *es un polinomio mónico (coeficiente de mayor grado unidad) con coeficientes en* F *e irreducible (no puede factorizarse en polinomios de menor grado con coeficientes en* F*) el anillo cociente* F[x]/(p(x)) *es un cuerpo, siendo* F[x] *el dominio integral de todos los polinomios con coeficientes en* F. *Cualquier cuerpo construido de esta manera se denomina* **extensión algebraica simple** *de* F.

Al llegar a este punto es preciso señalar que los elementos de $F[x]/(p(x))$ son las clases residuales de $(p(x))$; por tanto, cada una de ellas contiene exactamente un polinomio de grado menor que n, de la misma manera que la clase residual $(p)+n$ en Z contiene exactamente uno de los residuos 0, 1,..., p-1 que constituyen Z_p. Este tipo de construcción proporciona todos los cuerpos finitos.

A.3. CUERPOS FINITOS

Apoyándose en los resultados de la sección A.2, se llega a:

Teorema A.4: *Todo dominio integral finito es un cuerpo.*

Demostración: sea D un dominio integral finito con elementos no nulos $d_1, d_2, ..., d_n$. Para cualquier $x \neq 0$ en D se considera $xd_1, xd_2, ..., xd_n$ con $xd_i \neq 0$ dado que D es un dominio integral; lo que es más, $xd_i \neq xd_j$ si $i \neq j$, ya que si no, $d_i = d_j$; por tanto, debe existir algún d_j tal que $1 = xd_j$, con lo que $d_j = x^{-1}$, q.e.d.

Respecto a los cuerpos finitos, su propiedad más destacada se refiere a sus grupos multiplicativos, ya que son **cíclicos**; de este modo, es posible encontrar un elemento generador del grupo cíclico, g, que se denomina *elemento primitivo*; de este modo, para cada elemento x del grupo existe un entero positivo n (distinto para cada uno de ellos) tal que $x = g^n$.

Teorema A.5: *Para todo cuerpo finito existen un número primo* p *(la* característica *del cuerpo) y un entero positivo* n *(el grado del polinomio mínimo sobre* Z_p *de un elemento primitivo del cuerpo) de forma que el número de elementos del cuerpo es* p^n.

La demostración del teorema anterior es suficientemente compleja y será omitida aquí. Este resultado tiene un análogo para cualquier cuerpo F, de forma que éste contiene como subcuerpo primo (el subcuerpo que está contenido en cualquier otro subcuerpo de F) una copia de Z_p, para algún p primo; este valor de p también recibe, por analogía, el nombre de ***característica*** de F.

Es éste el momento de introducir la ampliamente usada notación GF(q), es decir, *cuerpo de Galois de orden* q, para denotar cualquier cuerpo finito de q elementos (por el teorema anterior, q debe ser potencia de un número primo); esta notación hace honor a

Evariste Galois (1811-1832), creador de los cuerpos finitos, y adelanta los resultados:

Corolario A.1: *Cualesquiera dos cuerpos con* p^n *elementos son isomórficos.*

Corolario A.2: *Cualquier cuerpo de característica* p *es una extensión algebraica simple de* Z_p*, o de otro modo, para cualquier cuerpo con* p^n *elementos existe un polinomio irreducible mónico* p(x) *de grado* n *tal que dicho cuerpo se identifica con* $Z_p[x]/(p(x))$.

Por ejemplo, $GF(16) \cong Z_2[x]/(x^4+x+1)$.

La importancia de este resultado deriva del hecho de que dos cuerpos finitos con el mismo número de elementos son formalmente equivalentes, lo que simplifica grandemente su estudio. Hasta ahora, la principal aplicación de los cuerpos finitos se centraba en el campo de la teoría de la información y la codificación, a través de los cuerpos de característica 2, $GF(2^n)$. En concreto, la información se transmite como una secuencia de unos y ceros, de forma que podemos identificar 1 y 0 con los elementos de GF(2), en virtud de los dos corolarios anteriores. Lo que es más, si consideramos los mensajes divididos en palabras de *n* bits, entonces cada una de las posibles palabras puede asociarse a un elemento de $GF(2^n)$. De este modo, uno de los códigos habituales para detección y corrección de errores, que transforma las palabras mensajes de *k* bits en palabras código de $n>k$ bits, puede interpretarse como una transformación desde $GF(2^k)$ a $GF(2^n)$. Por ejemplo, en los códigos lineales, esta transformación que hace corresponder un elemento de $GF(2^n)$ a cada elemento de $GF(2^k)$ puede representarse a través de una matriz, como ocurre para el código Hamming, que lleva los elementos de $GF(2^4)$ a $GF(2^7)$.

Sin embargo, existen códigos que hacen uso de propiedades específicas de los cuerpos finitos [LAC95, COH95] que consiguen una mejor aproximación a los límites establecidos por Shannon respecto a la relación entre la tasa de transmisión k/n y la fiabilidad del código; por ejemplo, los códigos de Reed-Solomon [SCH97], basados en GF(64) y GF(256), o la generalización de éstos propuestas por Goppa, matemático ruso que descubrió que por medio de curvas sobre cuerpos finitos se pueden obtener códigos eficaces (hablar de curvas en un cuerpo finito es algo abusivo, ya que el número de puntos

(x, y) que satisfacen la ecuación de la curva, $x^2+y^2=1$, por ejemplo, es finito; en este punto es preciso reseñar que todas las ecuaciones en un campo finito son polinómicas). De este modo pueden construirse códigos con distancia de Hamming 21, que tiene aplicación en transmisiones muy perturbadas, como las efectuadas por satélite. La principal limitación de estos códigos geométrico-algebraicos se refiere a los algoritmos de decodificación, que no eran excesivamente eficaces hasta fechas muy recientes [LAC95].

Sin embargo, no es la anterior la única aplicación que se puede dar a los cuerpos finitos, o al menos, a ciertas de sus propiedades en relación con Z_p, como la que sigue:

Propiedad A.1: *para cualquier cuerpo de Galois* GF(p) *todos sus elementos no nulos pueden generarse usando un* elemento primitivo g*; en concreto, existe un isomorfismo entre el grupo multiplicativo* $\{q_n\}=\{1, 2, ..., p-1\}$ *con la multiplicación módulo* p*, y el grupo aditivo* $\{i_n\}=\{0, 1, ...,p-2\}$ *con suma módulo* p-1*, de forma que para cada* q_n *existe un único* i_n *tal que:*

$$q_n = g^{i_n}$$

donde g *es el elemento primitivo de* GF(p).

Tal y como se ha visto, esta propiedad es una de las más usadas a fin de simplificar la multiplicación en cuerpos finitos.

APÉNDICE B

FAMILIA DE DISPOSITIVOS LÓGICOS PROGRAMABLES FLEX10K DE ALTERA

Este apéndice recoge las características básicas de la familia de CPLDs FLEX10K de Altera [ALT98], que se ha empleado a lo largo de este trabajo.

B.1. INTRODUCCIÓN

La familia FLEX10K constituye una de las primeras familias comerciales de CPLDs, capaz de albergar 10.000 puertas en los dispositivos más pequeños y hasta 250.000 puertas en los de mayor tamaño; además, los dispositivos FLEX10K son los primeros en el mercado que incluyen matrices embebidas, a través de los EABs, que proporcionan tablas de memoria para su uso como RAM o ROM sin necesidad de recurrir a los elementos lógicos convencionales. Esta familia está basada en bloques CMOS SRAM reconfigurables; la familia básica está fabricada en tecnología de 0.5μm, mientras que la familia FLEX10KA lo está en 0.35μm, y la FLEX10KE en 0.25μm; estos dispositivos operan a 5V, 3.3V y 2.5V, respectivamente.

Entre las principales características de la familia FLEX10K se pueden citar:

- posibilidad de multiplicación interna del reloj a través de la técnica ClockBoost;

- configuración a través de EPROM o controlador externo, o por puerto JTAG;
- compatible con el estándar PCI;
- consumo moderado (menos de 0.5 mA en modo de espera);
- cadenas de acarreo especiales para funciones aritméticas, y cadenas de cascada para funciones lógicas;
- emulación de buses internos triestado;
- hasta seis relojes y cuatro señales de clear globales;
- pines de entrada y salida triestado y en drenador abierto.

B.1. DESCRIPCIÓN FUNCIONAL

La figura B.1 muestra la estructura básica de la familia FLEX10K. Se puede apreciar que cada dispositivo está formado por una serie de bloques de elementos lógicos, o LABs, y bloques de memoria, o EABs; estos bloques están dispuestos en filas y columnas, con una columna central de EABs, y unidos entre sí por las conexiones programables, que a su vez, están conectadas a los elementos de E/S, o IOE.

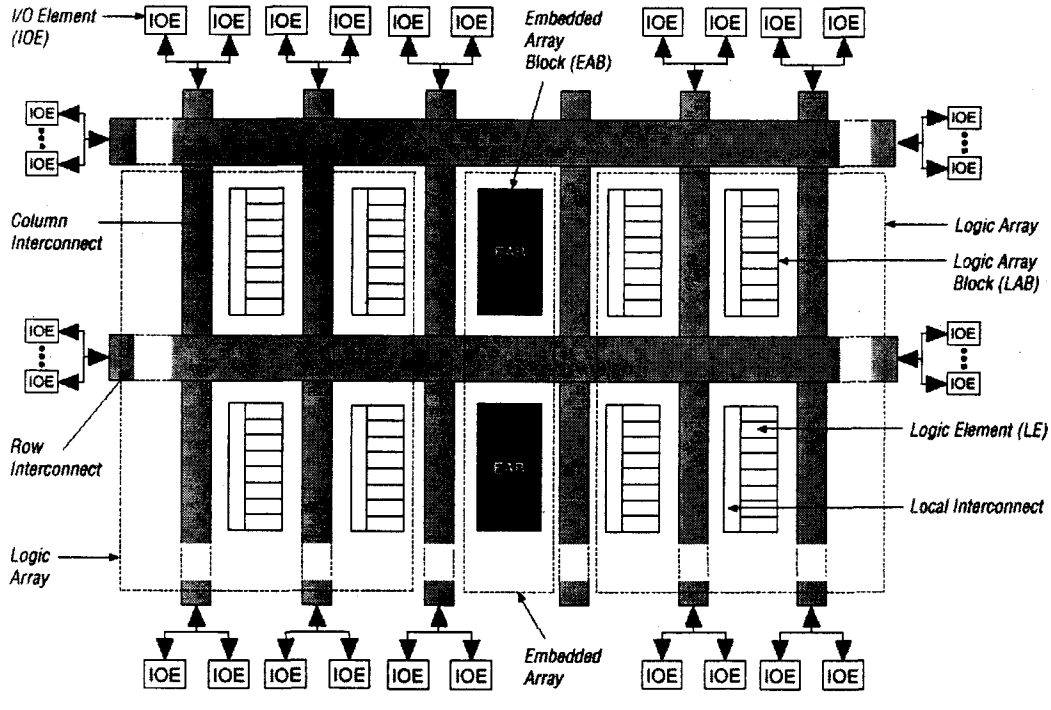


Figura B.1. Arquitectura de la familia FLEX10K.

La estructura de un LAB se muestra en la figura B.2, donde se puede apreciar que está formado por 8 elementos lógicos, o LEs, además de un bloque para la interconexión local entre los mismos y la conexión a los recursos globales de interconexión.

El LE es el bloque constitutivo de la familia FLEX10K, con chips albergando desde 576 LEs hasta 12160 LEs. Cada LE contiene una LUT de $2^4 \times 1$ bits, soporte para las cadenas de acarreo y de cascada, un registro programable y dos salidas. Una de las salidas está conectada a las interconexiones locales del LAB correspondiente, mientras que la otra comunica el LE directamente con las interconexiones rápidas globales de filas o columnas. Por su parte, las cadenas de acarreo y de cascada conectan todos los LEs en un mismo LAB, y todos los LABs en una misma fila, aunque el uso intensivo de estas últimas conexiones puede provocar problemas de enrutamiento. El soporte para la cadena de acarreo proporciona una propagación muy rápida (inferior a 0.3ns para dispositivos de 4ns

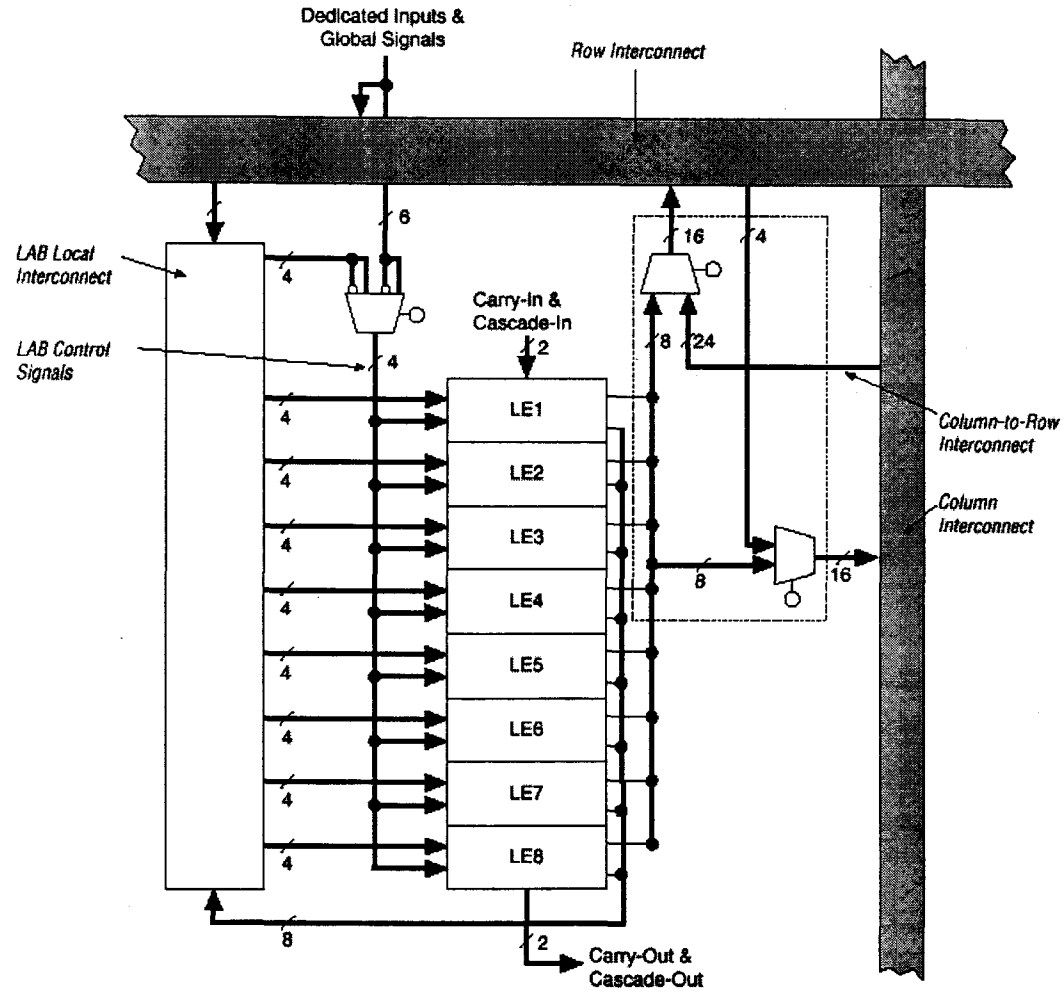


Figura B.2. Estructura de un LAB.

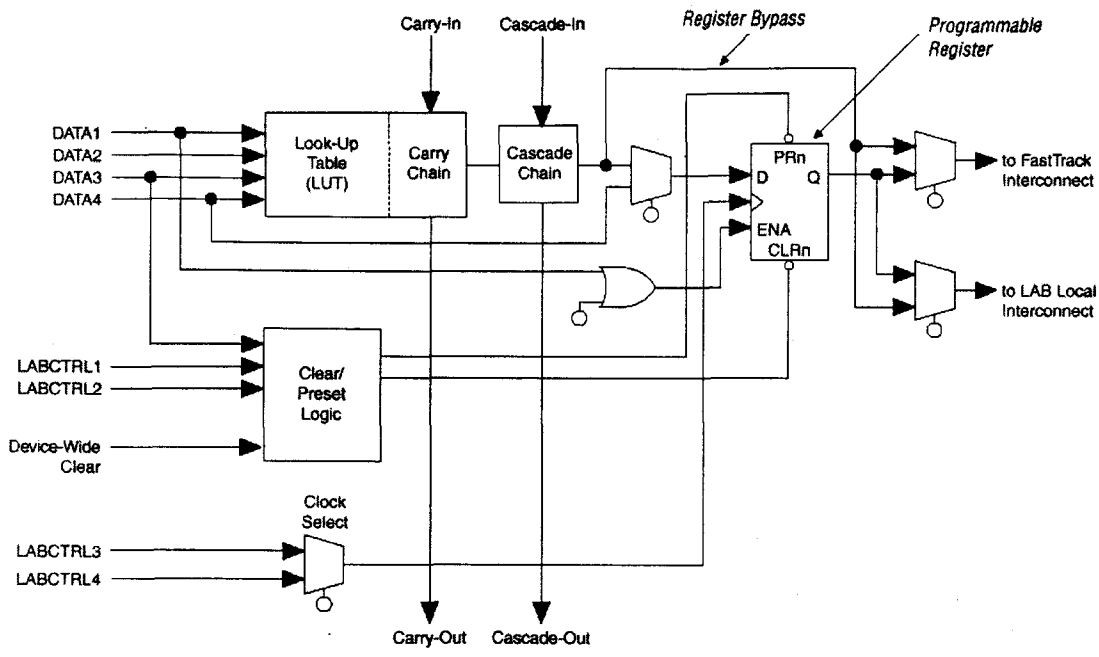


Figura B.3. Estructura de un LE.

de tiempo característico) de acarreos entre LEs, lo que permite implementar sumadores, contadores y comparadores de alta velocidad. Por su parte, la cadena de cascada se emplea para implementar funciones con gran fan-in, ya que permite combinar la salida de dos LEs adyacentes con una puerta AND o una puerta OR. La figura B.3 muestra la estructura básica de un LE.

Cada LE puede funcionar en cuatro modos diferentes: normal, aritmético, contador ascendente/descendente y contador reseteable; cada uno de estos modos usa los recursos de la figura B.3 de manera diferente. Puesto que se han empleado los dispositivos FLEX10K para la implementación de aplicaciones basadas en el RNS, el modo aritmético ha sido el más usado; así, la figura B.4 muestra la configuración de un LE funcionando en

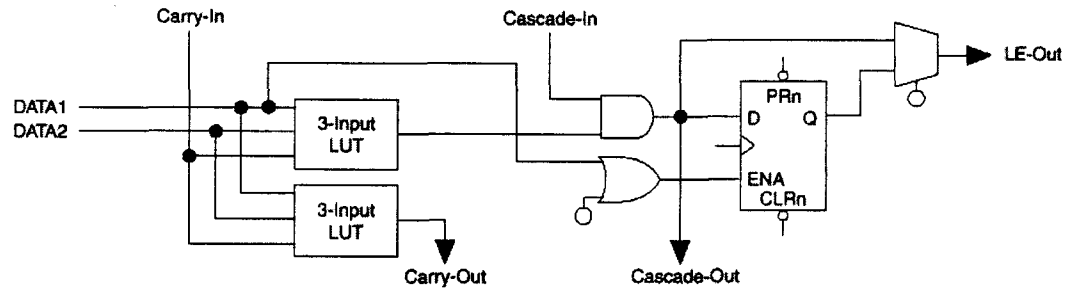


Figura B.4. LE en modo aritmético.

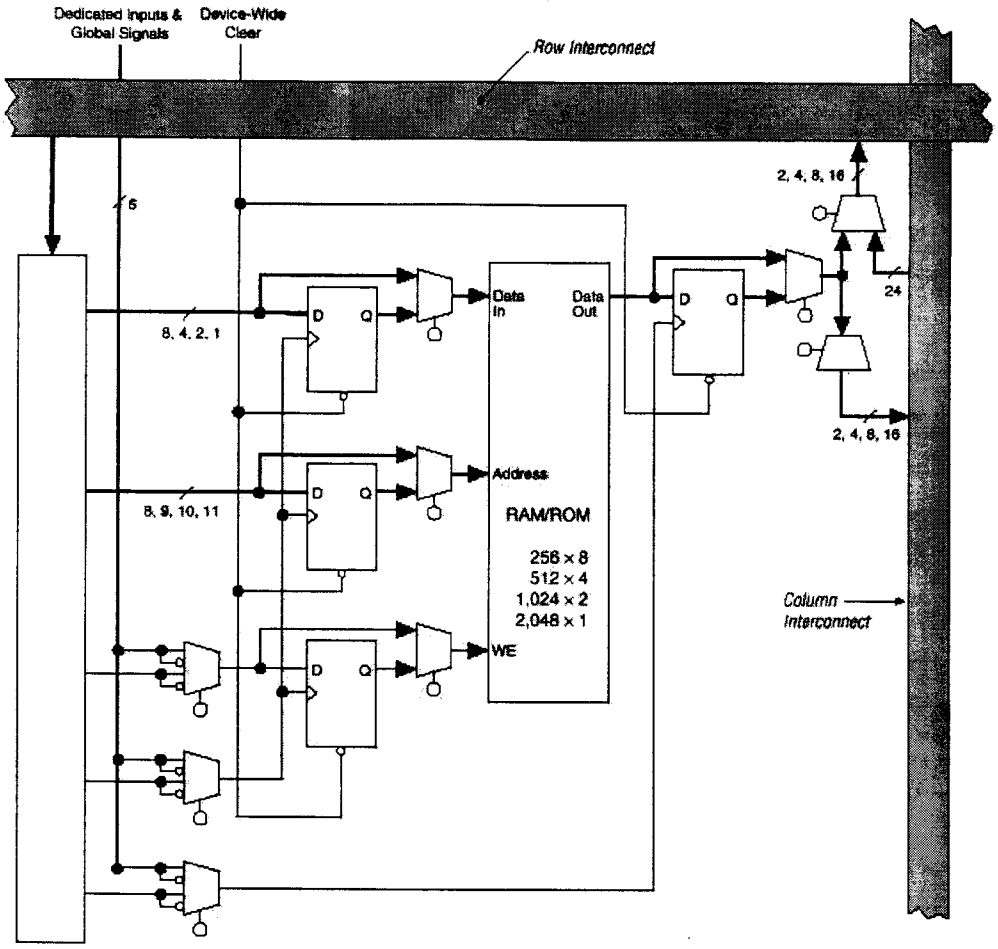


Figura B.5. Estructura de un EAB.

dicho modo. En este caso, la LUT del LE se reduce a $2^3 \times 1$ bits, mientras que el soporte de la cadena de acarreo, funcionando como otra LUT de $2^3 \times 1$ bits, genera la salida correspondiente para la cadena de acarreo; de esta manera, un LE trabajando en modo aritmético puede implementar un sumador de un bit de alta velocidad, con la posibilidad de hacer pasar la salida por el correspondiente registro.

Por otra parte, la figura B.5 muestra la estructura de un EAB. Cada EAB proporciona una tabla de $2^8 \times 8$ bits que puede funcionar como ROM o RAM y, en ambos casos, tanto las direcciones como los datos pueden funcionar síncrona o asíncronamente; además, cada EAB puede configurarse también como una tabla de $2^9 \times 4$, de $2^{10} \times 2$ ó de $2^{11} \times 1$ bits. De este modo, los EAB pueden emplearse tanto para la síntesis de pequeñas tablas de memoria RAM o ROM como para el almacenamiento de funciones complejas; por ejemplo, cada EAB de $2^8 \times 8$ bits puede albergar un producto parcial de 4×4 bits para

la implementación de multiplicadores binarios.

Por tanto, la familia FLEX10K de Altera reúne todos los recursos necesarios para la implementación eficiente de aplicaciones basadas en el RNS: soporte aritmético de alta velocidad y recursos específicos para la inclusión de pequeñas tablas de consulta; además, la propia estructura de los dispositivos, con LABs de ocho LEs, se adapta perfectamente al RNS, pudiendo integrar cada LAB una etapa básica en multitud de aplicaciones. De este modo, esta familia de CPLDs constituye una herramienta muy útil para la evaluación de arquitecturas RNS frente a las propuestas convencionales, al ser posible implementar ambas alternativas y medir sus prestaciones con la misma tecnología.

APÉNDICE C

PROCESO CMOS14TB

Este apéndice recoge las características fundamentales del proceso CMOS14TB, que se utilizó para la implementación VLSI del algoritmo de escalado basado en tablas de consulta propuesto en el capítulo V y para la simulación de la estrategia de sincronización propuesta en el capítulo VII.

C.1. INTRODUCCIÓN

El proceso de fabricación de circuitos integrados CMOS14TB es una adaptación del proceso HP-AMOS14TB de Hewlett-Packard para CMC (Canadian Microelectronics Corporation) a través del servicio MOSIS; de este modo, las estructuras VLSI expuestas en esta memoria han sido desarrolladas en las instalaciones del VLSI Research Group de la Universidad de Windsor, centro integrado en la red CMC. Dado que el uso de este proceso está sujeto a ciertos acuerdos de confidencialidad, sólo se va a realizar una breve descripción del mismo en este apéndice para ilustrar sus aspectos más destacados.

C.2. PRINCIPALES CARACTERÍSTICAS

El proceso CMOS14TB es un proceso para la fabricación de circuitos integrados digitales CMOS con 0.6μm de dimensión típica; sin embargo, la longitud efectiva del

canal para los transistores de dimensión mínima es 0.5μm. Por otra parte, el proceso está diseñado para circuitos con alimentación de 3.3V, aunque es posible la inclusión de elementos especiales de E/S para la adaptación entre 5V y 3.3V. El proceso CMOS14TB es un proceso de pozo n, con tres capas de metal y una de polisilicio. De este modo, el uso de tres capas de metal permite la realización de diseños complejos y relativamente compactos, dado que el espacio entre contactos corresponde a la dimensión mínima del proceso.

C.3. MODELOS DE SIMULACIÓN

Para finalizar, los modelos de SPICE y spectreS de los transistores utilizados para las diferentes simulaciones realizadas son los siguientes:

```
.MODEL CMOSN NMOS
+ PHI=0.700000 TOX=9.6000E-09 XJ=0.200000U TPG=1
+ VTO=0.6566 DELTA=6.9100E-01 LD=4.7290E-08 KP=1.9647E-04
+ UO=546.2 THETA=2.6840E-01 RSH=3.5120E+01 GAMMA=0.5976
+ NSUB=1.3920E+17 NFS=5.9090E+11 VMAX=2.0080E+05 ETA=3.7180E-02
+ KAPPA=2.8980E-02 CGDO=3.0515E-10 CGSO=3.0515E-10
+ CGBO=4.0239E-10 CJ=5.62E-04 MJ=0.559 CJSW=5.00E-11
+ MJSW=0.521 PB=0.99
+ XQC=0.5 XPART=0.5

.MODEL CMOSP PMOS
+ PHI=0.700000 TOX=9.6000E-09 XJ=0.200000U TPG=-1
+ VTO=-0.9213 DELTA=2.8750E-01 LD=3.5070E-08 KP=4.8740E-05
+ UO=135.5 THETA=1.8070E-01 RSH=1.1000E-01 GAMMA=0.4673
+ NSUB=8.5120E+16 NFS=6.5000E+11 VMAX=2.5420E+05 ETA=2.4500E-02
+ KAPPA=7.9580E+00 CGDO=2.3922E-10 CGSO=2.3922E-10
+ CGBO=3.7579E-10 CJ=9.35E-04 MJ=0.468 CJSW=2.89E-10
+ MJSW=0.505 PB=0.99
+ XQC=0.5 XPART=0.5
```

BIBLIOGRAFÍA

[ABR65] Abramowitz, M.; Stegun, I. A.: *Handbook of mathematical functions.* Dover Publications, 1965.

[AFG90] Afghahi, M.; Svensson, C.: *A unified single-phase clocking scheme for VLSI systems.* IEEE JOURNAL OF SOLID STATE CIRCUITS 1990, vol. 25, nº 1, págs. 225-233.

[ALI91] Alia, G.; Martinelli, E.: *A VLSI modulo m multiplier.* IEEE TRANSACTIONS ON COMPUTERS 1991, vol. 40, nº 7, págs. 873-878.

[ALT98] *Altera 1998 Data Book.* Altera Corporation, 1998.

[BAN74] Banerji, D. K.: *A novel implementation method for addition and substraction in residue number system.* IEEE TRANSACTIONS ON COMPUTERS 1974, vol. 23, nº 1, págs. 106-109.

[BAN81] Banerji, D. K.: *A high-speed division method in residue airthmetic.* Proc. of 1981 IEEE Fifth Symposium on Computer Arithmetic (mayo 1981), págs. 158-164.

[BAN94] Bandyopadhyay, S.; Jullien, G. A.; Sengupta, A.: *Fast VLSI systolic array for large modulus residue addition.* JOURNAL OF VLSI SIGNAL PROCESSING 1994, vol. 8, págs. 305-318.

[BAY87a] Bayoumi, M. A.; Jullien, G. A.; Miller, W. C.: *A VLSI implementation of residue adders*. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS 1987, vol. 34, págs. 284-288.

[BAY87b] Bayoumi, M. A.; Jullien, G. A.; Miller, W. C.: *A look-up table VLSI design methodology for RNS structures used in DSP applications*. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS 1987, vol. 34, págs. 604-613.

[BHA98] Bhardwaj, M.; Balaram, A.: *Low power signal processing architectures using residue arithmetic*. Proc. of 1998 IEEE International Conference on Acoustics, Speech and Signal Processing (Seattle, USA, 11-15 mayo), vol. 5, págs. 3017-3020. ISBN 0-7803-4428-6

[BOU95] Boussakta, S.; Holt, A. G. J.: *New transform using the Mersenne numbers*. IEE PROCEEDINGS-VISION, IMAGE AND SIGNAL PROCESSING 1995, vol. 142, nº 6, págs. 381-388.

[CHA97] Chang, K. C.: *Digital design and modeling with VHDL synthesis*. IEEE Computer Society Press, 1997.

[CHR95] Chren, W.A.: *RNS-based enhancements for direct digital frequency synthesis*. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II-ANALOG AND DIGITAL SIGNAL PROCESSING 1995, vol. 42, nº 8, págs. 516-524.

[COH95] Cohen, H.: *Números primos*. MUNDO CIENTIFICO 1995, vol. 161, págs. 846-851.

[ENC78] *Algebraic structures*. Encyclopaedia Britannica. Macropaedia, vol. 1, págs. 532-539. Encyclopaedia Britannica Inc. , 1978.

[ETZ80] Etzel, M.; Jenkins, W. K.: *Redundant residue number systems for error detection and correction in digital filters.* IEEE TRANSACTIONS ON ACOUSTICS, SPEECH AND SIGNAL PROESSING 1980, vol. 29, nº 5.

[GAR59] Garner, H. L.: *The Residue Number System.* IRE TRANSACTIONS ON ELECTRONIC COMPUTERS 1959, vol. 8, nº 6, págs. 140-147.

[GAR95] García Ríos, A.; Lloris Ruiz, A.: *Multiplicación en el sistema numérico de residuos.* Universidad de Granada, 1995. ISBN 84-7951-018-8

[GAR96] García Ríos, A.; Lloris Ruiz, A.: *Aritmética de residuos: una alternativa de cálculo.* MUNDO ELECTRÓNICO 1996, nº 272, págs 42-44. ISSN-0300-3787

[GAR97a] García Ríos, A.; Lloris Ruiz, A.: *Pipelined submodular isomorphic mapped RNS multipliers.* Proc. of XII Design of Circuits and Integrated Systems Conference (Sevilla, noviembre 1997), págs. 733-738. ISBN 84-88783-28-0

[GAR97b] García Ríos, A.; Lloris Ruiz, A.: *A scaling scheme based in conventional RNS arithmetical circuits.* Proc. of XII Design of Circuits and Integrated Systems Conference (Sevilla, noviembre 1997), págs. 739-744. ISBN 84-88783-28-0

[GAR97c] García, A.; Lloris, A.: *A look-up scheme for scaling in the RNS.* En proceso de segunda revisión en IEEE TRANSACTIONS ON COMPUTERS.

[GAR98a] García, A.; Meyer-Bäse, U.; Taylor, F. J.: *Pipelined Hogenauer CIC filters using field-programmable logic and residue number system.* Proc. of 1998 IEEE International Conference on Acoustics, Speech and Signal Processing (Seattle, USA, 11-15 mayo), vol. 5, págs. 3085-3088. ISBN 0-7803-4428-6

[GAR98b] García, A.; Lloris, A.: *RNS scaling based on pipelined multipliers for prime moduli.* Proc. of 1998 IEEE Workshop on Signal Processing Systems (Cambridge, USA, 8-10 octubre), págs. 459-468. ISBN 0-7803-4997-0

[GAR98c] García, A.; Lloris, A.: *Pipelined RNS multipliers: an application to scaling.* Proc. of 1998 IEEE International Conference on Electronics, Circuits and Systems (Lisboa, 7-10 septiembre), vol. 3, págs. 55-58. ISBN 0-7803-5008-1

[GAR98d] García, A.; Jullien, G. A.: *Comments on "An arithmetic free parallel mixed-radix conversion algorithm".* En proceso de revisión en IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II-ANALOG AND DIGITAL SIGNAL PROCESSING.

[GAR99] García, A.; Meyer-Bäse, U.; Lloris, A.; Taylor, F. J.: *RNS implementaion of FIR filters based on distributed arithmetic using field-programmable logic.* Aceptado en 1999 IEEE International Conference on Circuits and Systems, (Orlando, USA, 30 mayo - 2 junio).

[GOS94] Goslin, G.; Newgard, B.: *16-tap, 8-bit FIR filter applications guide.* Xilinx Application note, 1994. http://www. xilinx. com/appnotes/fir_filt. pdf

[GOS95] Goslin, G. R.: *A guide to using Field Programmable Gate Arrays (FPGAs) for application-specific digital signal processing performance.* Xilinx Application note, 1995. http://www. xilinx. com/appnotes/dspguide. pdf

[GRI88] Griffin, M.; Taylor, F. J.; Sousa, M.: *New scaling algorithms for the chinese remainder theorem.* Proc. of 22nd Asilomar Conference on Signals, Systems and Computers, 1988.

[GRI89] Griffin, M.; Sousa, M.; Taylor, F. J.: *Efficient scaling in the residue number system.* Proc. of 1989 International Conference on Acoustics, Speech and Signal Processing, págs. 1075-1078.

[HIA97] Hiasat, A.; Abdel-Aty-Zohdy, H.S.: *Design and implementation of an RNS division algorithm.* Proc. of 13th IEEE Symposium on Computer Arithmetic (Asilomar, USA, julio 1997), págs. 240-249.

[HIT95] Hitz, M. A.; Kaltofen, E.: *Integer division in residue number system.* IEEE TRANSACTIONS ON COMPUTERS 1995, vol. 44, nº 8, págs. 983-989.

[HOG81] Hogenauer, E. B.: *An economical class of digital filters for decimation and interpolation.* IEEE TRANSACTIONS ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING 1981, vol. 29, nº 2, págs. 155-162.

[HUA81] Huang, C. H.; Peterson, D. G.; Rauch, H. E.; Teague, J. W.; Fraser, D. F.: *Implementation of a fast digital processor using the residue number system.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS 1981, vol. 28, nº 1, págs. 32-38.

[HUA83] Huang, C. H.: *A fully parallel mixed-radix conversion algorithm for residue number applications.* IEEE TRANSACTIONS ON COMPUTERS 1983, vol. 32, págs. 398-402.

[IBR94] Ibrahim, K. M.: *Novel digital filter implementation using hybrid RNS-binary arithmetic.* SIGNAL PROCESSING 1994, vol. 40, págs. 287-294.

[JEN77] Jenkins, W. K.; Leon, B. J.: *The use of residue number systems in the design of finite impulse response filters.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS 1977, vol. 24, nº 4, págs. 191-201.

[JEN83] Jenkins, W. K.; Krozmeier, J. J.: *Error detection and correction in quadratic residue number systems.* Proc. of 26th Midwest Symposium on Circuits and Systems (Pueblo Mexico, USA, agosto 1983).

[JIR87] Ji-Ren, Y.; Karlsson, I.; Svensson, C.: *A true single-phase-clock dynamic CMOS circuit technique*. IEEE JOURNAL OF SOLID STATE CIRCUITS 1987, vol. 22, nº 5, págs. 899-901.

[JUL78] Jullien, G. A.: *Residue number scaling and other operations using ROM arrays*. IEEE TRANSACTIONS ON COMPUTERS 1978, vol. 27, nº 4, págs. 325-357.

[JUL80] Jullien, G. A.: *Implementation of multiplication, modulo a prime number, with applications*. IEEE TRANSACTIONS ON COMPUTERS 1980, vol. 29, nº 10, págs. 899-905.

[JUL88] Jullien, G. A.; Taheri, M.; Miller, W. C.: *A VLSI radix-2 two-dimensional FFT butterfly with fault tolerance*. 1988 Conference on Computers, invited paper, (Baton Rouge, USA), octubre 1998.

[KAM79] Kameyama, M.; Higuchi, T.: *A new scaling algorithm in symmetric residue number system based on multiple-valued logic*. Proc. IEEE International Symposium on Circuits and Systems (Tokyo, Japan), págs. 189-192, Julio 1979.

[KAM89] Kameyama, M.; Sekibe, T.; Higuchi, T.: *Highly parallel residue arithmetic chip based on multiple-valued bidirectional current-mode logic*. IEEE JOURNAL OF SOLID-STATE CIRCUITS 1989, vol. 24, nº 5, págs. 1404-1411.

[KAT96] Katti, S.: *A new residue arithmetic error correction scheme*. IEEE TRANSACTIONS ON COMPUTERS 1996, vol. 45, nº 1, págs. 13-19.

[KIN74] Kinoshita, E.; Kosako, H.; Kojima, Y.: *Floating-point arithmetic algorithms in the symmetric residue number system*. IEEE TRANSACTIONS ON COMPUTERS 1974, vol. 23, nº 1, págs. 9-20.

[KIN97] Kinoshita, E.; Lee, K.-J.: *A residue arithmetic extension for reliable scientific computation.* IEEE TRANSACTIONS ON COMPUTERS 1997, vol. 46, nº 2, págs. 129-138.

[LAC95] Lachaud, G.; Vladut, S.: *Códigos correctores de errores.* MUNDO CIENTIFICO 1995, vol. 161, págs. 864-868.

[LAK94] Lakhani, G.: *Some fast residual arithmetic adders.* INTERNATIONAL JOURNAL OF ELECTRONICS 1994, vol. 77, nº 2, págs. 225-240.

[LIP81] Lipson, J. D.: *Elements of algebra and algebraic computing.* Addison-Wesley, 1981.

[MAN72] Mandelbaun, D.: *Error correction in residue arithmetic.* IEEE TRANSACTIONS ON COMPUTERS 1972, vol. 21, nº 6, págs. 538-545.

[MIL98] Miller, D. F.; McCormick, W. S.: *An arithmetic free parallel mixed-radix conversion algorithm.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II-ANALOG AND DIGITAL SIGNAL PROCESSING 1998, vol. 45, nº 1, págs. 158-162.

[ORT92] Orton, G. A.; Peppard, L. E.; Tavares, S. E.: *New fault tolerant techniques for residue number systems.* IEEE TRANSACTIONS ON COMPUTERS 1992, vol. 41, nº 11, págs. 1453-1464.

[PEL74] Peled, A.; Liu, B.: *A new hardware realization of digital filters.* IEEE TRANSACTIONS ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING 1974, vol. 22, págs. 456-462.

[POS96] Posch, K. C.; Posch, R.: *Division in residue number systems involving length indicators.* JOURNAL OF COMPUTATIONAL AND APPLIED MATHEMATICS 1996, vol. 66, págs. 411-419.

[PRI86] Pries, W.; Thanailakis, A.; Card, H. C.: *Group properties of cellular automata and VLSI applications.* IEEE TRANSACTIONS ON COMPUTERS 1986, vol. 35, nº 12, págs. 1013-1042.

[RAD91] Radhakrishnan, D.; Yuan, Y.: *Fast and highly compact RNS multipliers.* INTERNATIONAL JOURNAL OF ELECTRONICS 1991, vol. 70, nº 2, págs. 281-293.

[RAD98] Radhakrishnan, D.; Preethy, A.: *A new approach to data conversion: direct analog-to-residue converter.* Proc. of 1998 IEEE International Conference on Acoustics, Speech and Signal Processing (Seattle, USA, 11-15 mayo), vol. 5, págs. 3013-3016. ISBN 0-7803-4428-6

[SCH97] Schroeder, M. R.: *Number thoery in science and communication.* Springer Verlag, 1997.

[SMI95] Smith, J.C.; Taylor, F.J.: *A fault-tolerant GEQRNS processing element for linear systolic array DSP applications.* IEEE TRANSACTIONS ON COMPUTERS 1995, vol. 44, nº 9, págs. 1121-1130.

[SOD86] Soderstrand, M. A.; Jenkins, W. K.; Jullien, G. A.; Taylor, F. J.: *Residue number system arithmetic: modern applications in digital signal processing.* IEEE Press, New York, 1986.

[SU90] Su, C. C.; Lo, H. Y.: *An algorithm for scaling and single residue error correction in the residue number system.* IEEE TRANSACTIONS ON COMPUTERS 1990, vol. 39, nº 8, págs. 1053-1064.

[SZA67] Szabo, N. S.; Tanaka, R. I.: *Residue arithmetic and its applications to computer technology.* McGraw-Hill, New York, 1967.

[TAH88] Taheri, M.; Jullien, G. A.; Miller. W. C.: *High speed signal processing using systolic arrays over finite rings.* IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS 1988, vol. 6, nº 3, págs. 504-512.

[TAT95] Tatsaki, A.; Stouraitis, T.; Goutis, C.: *Image coder based on residue number system for progressive transmission.* ELECTRONICS LETTERS 1995, vol. 31, nº 6, págs. 442-443.

[TAY82] Taylor, F. J.: *A VLSI residue arithmetic multiplier.* IEEE TRANSACTIONS ON COMPUTERS 1982, vol. 31, nº 6, págs. 540-546.

[TAY84] Taylor, F. J.: *Residue arithmetic: a tutorial with examples.* IEEE COMPUTER 1984, vol. 17, nº 5, págs. 50-62.

[TAY86] Taylor, F. J.: *An analysis of the distributed arithmetic digital filter.* IEEE TRANSACTIONS ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING 1986, vol. 34, nº 5, págs. 1165-1170.

[TAY98] Taylor, F. J.; Mellott, J.: *Hands-on digital signal processing.* McGraw-Hill, 1998.

[TSE79] Tseng, B.; Jullien, G. A.; Miller, W. C.: *Implementation of FFT structures using the residue number system.* IEEE TRANSACTIONS ON COMPUTERS 1979, vol. 28, nº 11, págs. 831-845.

[UYE92] Uyemura, J. D.: *Circuit design for CMOS VLSI.* Kluwer Academic Publishers, 1992.

[WAN94] Wang, C.-L.: *New bit-serial VLSI implementation of RNS FIR digital filters.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II-ANALOG AND DIGITAL SIGNAL PROCESSING 1994, vol. 41, nº 11, págs. 768-772.

[WAN96] Wang, Z; Jullien, G. A.; Miller, W. C.: *Efficient tree architecture for modulo 2^n+1 multiplication.* JOURNAL OF VLSI SIGNAL PROCESSING, vol. 14, nº 3, págs. 241-248.

[WAS82] Waser, S.; Flynn, M.J.: *Introduction to arithmetic for digital systems designers.* Holt, Rinehart & Winston, 1982.

[WHI89] White, S. A.: *Applications of distributed arithmetic to digital signal processing: a tutorial review.* IEEE ACOUSTICS, SPEECH AND SIGNAL PROCESSING MAGAZINE 1989, págs. 4-19.

[WIG90] Wigley, N. M.; Jullien, G. A.: *On moduli replication for residue arithmetic computations of complex inner products.* IEEE TRANSACTIONS ON COMPUTERS 1990, vol. 39, págs. 1065-1076.

[WIG94] Wigley, N. M.; Jullien, G. A.; Reaume, D.: *Large dynamic range computations over small finite rings.* IEEE TRANSACTIONS ON COMPUTERS 1994, vol. 43, nº 1, págs. 78-86.

[YOR91] York, T. A.; Srisuchinwong, B.; Tsalides, P.; Hicks, P. J.; Thanailakis, A.: *Design and VLSI implementation of mod-127 multiplier using cellular automaton-based data compression techniques.* IEE PROCEEDINGS G-CIRCUITS, DEVICES AND SYSTEMS 1991, vol. 138, nº 5, págs. 351-356.

[YUA89] Yuan, J.; Svensson, C.: *High-speed CMOS circuit technique.* IEEE JOURNAL OF SOLID STATE CIRCUITS 1989, vol. 24, nº 1, págs. 62-70.

[ZEL91] Zelniker, G.; Taylor, F. J.: *A reduced-complexity finite field ALU.* IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS 1991, vol. 38, nº 12, págs, 1571-1573.

[ZHA90] Zhang, D; Jullien, G. A.; Miller, W. C.: *A neural-like network approach to finite ring computations*. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS 1990, vol. 37, nº 8, págs, 1048-1051.